Jül-4309

Jülich Supercomputing Centre (JSC)

Automatisierung und Optimierung der Auswertung wissenschaftlicher Experimentdaten

Jochen Kreutz

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Automatisierung und Optimierung der Auswertung wissenschaftlicher Experimentdaten

Jochen Kreutz

Berichte des Forschungszentrums Jülich; 4309
ISSN 0944-2952
Jülich Supercomputing Centre (JSC)
Jül-4309

Vollständig frei verfügbar im Internet auf dem Jülicher Open Access Server (JUWEL) unter
http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Title:

Automation and Optimization of the Analysis of Data from Scientific Experiments

Abstract:

Scientific experiments often generate huge amounts of data that must be processed and evaluated. Data processing should therefore be automated to as great an extent as possible and standardized central access to the data should be guaranteed. An example of such a scientific experiment is one with the TEXTOR fusion reaction operated at Forschungszentrum Jülich. TEXTOR is used to investigate plasma-wall interactions in nuclear fusion. Due to the many different measuring techniques used, huge amounts of experimental data are generated, all of which must be stored centrally and subsequently analyzed. For the administration and analysis of the physical signals calculated from the experimental data, the TEXTOR Physical Database (TPD) is used. Here, all of the calculated versions of a physical signal are saved.

Some of the physical signals in TPD depend on each other. In order to ensure that dependent signals are also updated when a new signal version is created, dependent TPD signals are automatically recalculated on the basis of database triggers and PL/SQL procedures. This involves disintegrating the dependencies using existing metainformation and by conducting a depth-first search. In addition to the recalculation of dependent signals, metainformation on the automatically generated signal versions is simultaneously generated and added to the TPD.

As the query times of TPD signals play an important role in evaluating the experiment, the potential for improving the performance of database queries was also examined. In doing so, signal tables were partitioned and the use of materialized views considered.

Automatisierung und Optimierung der Auswertung wissenschaftlicher Experimentdaten

Wissenschaftliche Experimente erzeugen häufig große Mengen an Daten, die verarbeitet und ausgewertet werden müssen. Die Datenverarbeitung sollte dabei soweit wie möglich automatisiert und ein einheitlicher zentraler Zugriff auf die Daten bereitgestellt werden.
Ein Beispiel für ein solches wissenschaftliches Experiment ist der im Forschungszentrum Jülich betriebene Fusionsreaktor TEXTOR, mit dessen Hilfe Plasma-Wandwechselwirkungen bei der Kernfusion untersucht werden.
Auf Grund der vielen verwendeten Diagnostiken werden hier große Mengen von Messdaten erzeugt, die zentral gespeichert und anschließend analysiert werden müssen. Für die Verwaltung und Auswertung der aus den Messdaten berechneten physikalischen Signale wird die TEXTOR Physical Database (TPD) bereitgestellt, in der sämtliche berechnete Versionen eines physikalischen Signals gespeichert sind.

Die physikalischen Signale in der TPD sind zum Teil voneinander abhängig. Damit bei der Erstellung einer neuen Signalversion auch die abhängigen Signale aktualisiert werden, wurde eine automatische Neuberechnung abhängiger TPD-Signale auf Basis von Datenbank-Triggern und PL/SQL-Prozeduren implementiert. Die Abhängigkeiten werden dabei durch die Verwendung vorhandener Metainformationen und das Durchführen einer Tiefensuche aufgelöst. Neben der Neuberechnung abhängiger Signale werden gleichzeitig Metainformationen über die automatisch erzeugten Signalversionen erzeugt und in die TPD eingefügt.

Da die Abfragezeiten von TPD-Signalen eine wichtige Rolle bei der Auswertung des Experiments spielen, wurden zudem mögliche Performance-Verbesserungen der Datenbankabfragen untersucht. Hierbei wurden unter anderem die Partitionierung von Signaltabellen und der Einsatz von Materialized Views betrachtet.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Bei der Durchführung vieler wissenschaftlicher Experimenten fallen sehr große Datenmengen an. Für die Verarbeitung der Experimentdaten gibt es einige Herausforderungen. Auf Grund der großen Anzahl von Daten sollte die Datenverarbeitung soweit wie möglich automatisiert werden. Dabei sind die Datensicherheit durch geeignete Maßnahmen, wie z.B. regelmäßige Sicherungen, sowie die Transparenz und Flexibilität der einzelnen Vorgänge zu berücksichtigen. In der Regel müssen die aufgezeichneten Daten eines Experiments durch entsprechende Programme weiterverarbeitet werden, um daraus physikalische Größen zu erzeugen. Diese berechneten Größen sowie die zu Grunde liegenden Programme können sich je nach Erkenntnissen der Auswertung unter Umständen im Laufe der Zeit ändern. Für die Auswertung der Experimentdaten sollte zudem ein zentraler und einheitlicher Zugriff auf die Daten bereitgestellt werden, der unabhängig von den evtl. auftretenden verschiedenen Datentypen möglichst intuitiv verwendet werden kann.
Ein Beispiel für ein solches wissenschaftliches Experiment ist der Fusionsreaktor TEXTOR. Dieses Großgerät wird durch das Institut für Plasmaphysik (IEF-4) im Forschungszentrum Jülich GmbH zur Untersuchung von Plasma-Instabilitäten und Plasma-Wand-Wechselwirkungen bei der Kernfusion betrieben.
Auf Grund der zahlreichen eingesetzten Diagnostiken fallen bei den im Rahmen der Untersuchung durchgeführten Messungen sehr große Datenmengen an, die analysiert und ausgewertet werden müssen. Hierbei werden die von den Messapparaturen erzeugten Rohdaten zunächst auf einem zentralen Fileserver gespeichert. Im Anschluss an eine Messung werden automatisch Programme gestartet, die aus den Rohdaten weitere physikalische Signaldaten berechnen und diese in einer Datenbank ablegen. Häufig werden im Nachhinein weitere Signalberechnungen durchgeführt und so mehrere Versionen verschiedener Signale (z.B. durch Verwendung unterschiedlicher Parameter in den Programmen) erzeugt. Auf Grund von Abhängigkeiten zwischen den physikalischen Signalen sollte gewährleistet sein, dass abhängige Signale stets in einer aktuellen Version vorliegen. Das heißt, sobald eine neue Signalversion erzeugt wird, müssen auch die von diesem Signal abhängigen Signale neu berechnet werden.

Im Rahmen dieser Arbeit soll daher eine automatische Berechnung abhängiger Signale implementiert werden, die die Konsistenz der Versionen abhängiger Signale gewährleistet. Um die Suche nach Signalen und die Analyse der Daten zu vereinfachen, sollen zudem Metainformationen, wie z.B. Zeitpunkt und Anlass für eine Signalberechnung, gespeichert und eine Weboberfläche erstellt werden, mit der sich die Metainformationen darstellen lassen.
Der Zugriff auf die in der Datenbank abgelegten Signaldaten spielt sowohl bei der Signalberechnung als auch bei der Analyse und Auswertung der Daten eine wichtige Rolle. Ein effektives Arbeiten mit den Daten setzt einen schnellen Zugriff auf die Datenbank voraus. Ein weiterer Aspekt der Arbeit ist daher, die Performance der Datenbankabfragen durch den Einsatz geeigneter Maßnahmen zu optimieren. Hierbei sollen unter anderem das Partitionieren der Signaltabellen und eine mögliche Verwendung von Materialized Views untersucht werden.

Kapitel 2

Das TEXTOR-Experiment

Im Folgenden wird das TEXTOR-Experiment beschrieben. Dabei werden die physikalischen Hintergründe der Kernfusion und die mögliche Nutzung als Energiequelle kurz vorgestellt. Anschließend wird das TOKAMAK-Prinzip erklärt und die Eigenschaften des Versuchsaufbaus beschrieben.

2.1 Energiegewinnung durch Kernfusion

Zunächst sind die physikalischen Abläufe und die Grundlagen der Kernfusion einzuführen, bevor auf die mögliche Nutzung der Kernfusion als Energiequelle und die dabei zu bewältigenden Problemstellungen eingegangen wird.

2.1.1 Prinzip der Kernfusion

Die Kernfusion wird als zukünftige Energiequelle für die langfristige Energieversorgung gesehen. Unter der Kernfusion versteht man den Vorgang, bei dem leichte Atomkerne (z.B. Wasserstoffkerne) miteinander verschmelzen und damit schwerere Kerne bilden. Dieser Vorgang geschieht unter starkem Druck und hoher Temperatur und setzt eine große Menge Energie frei. Solche Prozesse finden im Inneren von Sternen statt und versorgen sie mit Energie. Dort verschmelzen Wasserstoffkerne bei einem Druck von ca. $2 * 10^{11}$ bar und einer Temperatur von ca. 15 Millionen Grad. Im Vergleich zu einer chemischen Reaktion besitzt die Kernfusion ein wesentlich höheres Energiepotential. Aus einem einzigen Gramm der Wasserstoffisotope Deuterium und Tritium lassen sich beispielsweise durch Kernfusion ca. $3,4 * 10^{11}$ Joule Energie freisetzen. Das entspricht einem Energiegehalt von 10.000 Litern Benzin [LIT01, S.140].
Weitere Vorteile der Kernfusion sind die gleichmäßige großflächige Verfügbarkeit der Rohstoffe, die zur Brennstofferzeugung benötigt werden, sowie die Tatsache, dass bei der Kernfusion keine umweltschädlichen Treibhausgase entstehen. Zwar werden ähnlich

wie bei der Kernspaltung auch bei der Kernfusion in der Regel radioaktive Materialien verwendet, jedoch ist deren Handhabung und Entsorgung auf Grund geringer Halbwertszeiten weniger kritisch.

Im Gegensatz zu bekannten Verbrennungsreaktionen, wie z.B. von Kohlenstoff und Sauerstoff zu Kohlenstoffdioxid, bei denen die chemischen Reaktionen durch die Elektronenhüllen der Atome verursacht werden, beruht die Kernfusion auf den Kräften zwischen den Atomkernen. Da diese Kräfte sehr groß sind, wird bei der Kernfusion ca. 10^6 mal soviel Energie freigesetzt wie bei chemischen Verbrennungsreaktionen. Die Energieabgabe wird dadurch ermöglicht, dass die Masse eines Atomkerns stets kleiner als die Summe der Massen seiner Bausteine ist. Der Betrag der freigesetzten Energie entspricht wegen der Äquivalenz von Masse und Energie:

$$\Delta E = \Delta m c^2$$

Der größte Teil der freiwerdenden Energie versorgt die Reaktionsprodukte mit kinetischer Energie ([LIT02, S. 9]). Bei der Kernfusion werden die Wasserstoffisotope Deuterium (1 Proton, 1 Neutron) und Tritium (1 Proton, 2 Neutronen) zu Helium verschmolzen. Abbildung 2.1 zeigt die Verschmelzung von Deuterium und Tritium zu einem Heliumatom und einem zusätzlich freiwerdenden Neutron.

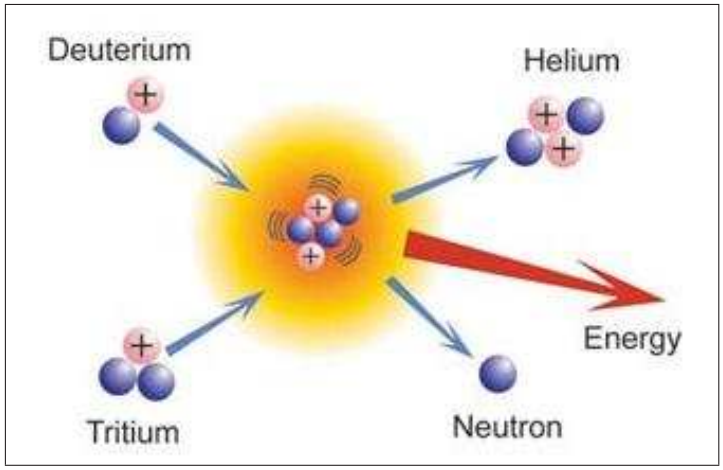


Abbildung 2.1: Reaktion von Deuterium und Tritium
(Quelle: www.iter.org)

Da Deuterium in Wasser enthalten ist und sich Tritium durch relativ geringen Aufwand aus Lithium erzeugen lässt, welches großflächig in Gestein auf der ganzen Erde vorkommt, könnte die Kernfusion eine Energiequelle für tausende von Jahren darstellen.

2.1.2 Nutzung als Energiequelle

Weil die Reaktion der Atomkerne nur unter großem Druck und hohen Temperaturen stattfindet, muss der Brennstoff extrem stark erhitzt werden, um den Fusionvorgang anzustossen. Dabei entsteht ein sogenanntes Plasma, in dem sich die Atomkerne und Elektronen nahezu frei bewegen können. Für das Verschmelzen von Deuterium und Tritium sind ein Druck von ca. 5 bis 10 bar und eine Temperatur von ca. 100 Millionen Grad nötig [LIT01, Seite 140]. Um das Plasma auf die notwendige Temperatur zu bringen, sind spezielle Heizmethoden und viel Energie erforderlich.

Ein Problem bei der Nutzung der Kernfusion als Energiequelle ist der Einschluss des Plasmas. Da es keinen Werkstoff gibt, der bei den auftretenden enorm hohen Temperaturen als Druckbehälter dienen könnte, muss das Plasma deshalb von den Wänden des Behälters ferngehalten werden. Die gewünschte Reaktion der Teilchen kann zudem nur stattfinden, wenn diese sich nah genug beieinander befinden.
Ein Ansatz zur Eingrenzung des Plasmas ist der sogenannte ”Trägheitseinschluss”. Dieser nutzt die Verzögerung der Expansion des Brennstoffs aus. Der Brennstoff muss dabei so schnell erhitzt werden, dass vor der Ausdehnung des Plasmas bereits ausreichend viele Reaktionen zwischen den Atomkernen stattgefunden haben. Die Anwendbarkeit dieser Methode bleibt jedoch experimentel zu belegen, da derzeit die notwendigen Voraussetzungen (z.B. bezüglich der Heizleistung) nicht erfüllt werden können. [LIT02]
Die bisher am weitesten verbreitetste Methode zum Einschluss des Plasmas ist die Nutzung starker Magnetfelder in einem Vakuumbehälter. Hierbei wird versucht, die Teilchen im Plasma möglichst lange auf identischen Bahnen innerhalb des ausgedehnten Plasmas zu halten. Grundlage hierfür ist die Tatsache, dass geladene Teilchen sich durch die Lorentzkraft spiralförmig um magnetische Feldlinien bewegen (vgl. Abbildung 2.2).

Erste Ideen zu Fusionsreaktoren kamen bereits nach dem Zweiten Weltkrieg auf. Ein Ansatz für den Einschluss des Plasmas durch Magnetfelder ist der sogenannte ”Stellarator”. Hierbei werden die Magnetfelder komplett durch Magnetspulen außerhalb des Behälters erzeugt. Die Herstellung der speziellen Magnetspulen sowie deren Anordnung sind jedoch sehr komplex und aufwendig.
In den sechziger Jahren hat sich der in Moskau entwickelte TOKAMAK (”Toroidale Kammer in Magnetspulen”) durchgesetzt und ist die derzeit am häufigsten eingesetzte Technik. Grundlage hierfür ist ebenfalls das zuvor beschriebene Verhalten geladener Teilchen in einem Magnetfeld. Das TOKAMAK-Prinzip wird in Kapitel 2.2 näher beschrieben.

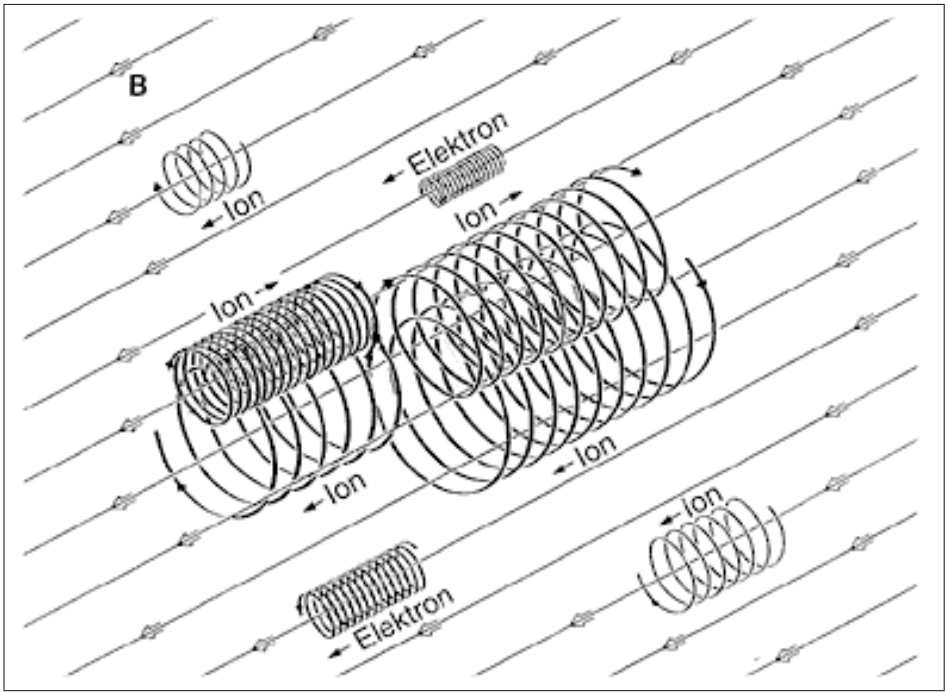


Abbildung 2.2: Bewegung geladener Teilchen in einem Magnetfeld

Selbst wenn genügend Reaktionen im Plasma herbeigeführt werden können, muss die freigewordene Energie zurückgewonnen und nutzbar gemacht werden. Wie bereits erwähnt, wird der größte Teil der Energie in Form energiereicher Neutronen (kinetische Energie) freigesetzt. Diese lässt sich in thermische Energie umwandeln, welche wiederum mit Hilfe gängiger Verfahren, wie etwa das Antreiben von Turbinen, in elektrische Energie umgewandelt werden kann [LIT03].

Trotz großer Erfolge in den letzten Jahrzehnten sind noch viele technische Entwicklungen notwendig, um die Kernfusion als Energiequelle nutzbar zu machen.

2.2 Aufbau eines TOKAMAK-Fusionsreaktors

Da der TOKAMAK-Aufbau eine wichtige Rolle in der Kernfusion und für das TEXTOR-Experiment spielt, werden dessen Eigenschaften zunächst im Allgemeinen und dann am Beispiel von TEXTOR erklärt.

2.2.1 Das TOKAMAK-Prinzip

Das Plasma ist instabil bezüglich der Form und Lage. Um diese Instabilitäten in den Griff zu bekommen, ist ein einziges Magnetfeld nicht ausreichend. Erst durch Verwendung mehrerer überlagerter Magnetfelder ist die Lageregelung des Plasmas beherrschbar. Der TOKAMAK besitzt daher zwei zusammenwirkende Magnetfelder: ein toroidales und ein poloidales. Das toroidale Magnetfeld wird durch Magnetspulen erzeugt, die um den Torus gewickelt werden. Das poloidale Feld wird mit einem durch das Plasma fließenden Strom erzeugt, der durch einen Transformator induziert wird. Beide Felder ergänzen sich zu einem Magnetfeld, welches den Torus spiralförmig umgibt und das Plasma von den Wänden des Torus fernhält. Der im Plasma induzierte Strom kann gleichzeitig als Heizquelle verwendet werden.
Da die Induktion durch monotone Stromänderungen erfolgt, ist ein TOKAMAK ein gepulster Aufbau, der nicht kontinuierlich betrieben werden kann. Abbildung 2.3 verdeutlicht die Anordnung der Magnetfelder eines TOKAMAK.

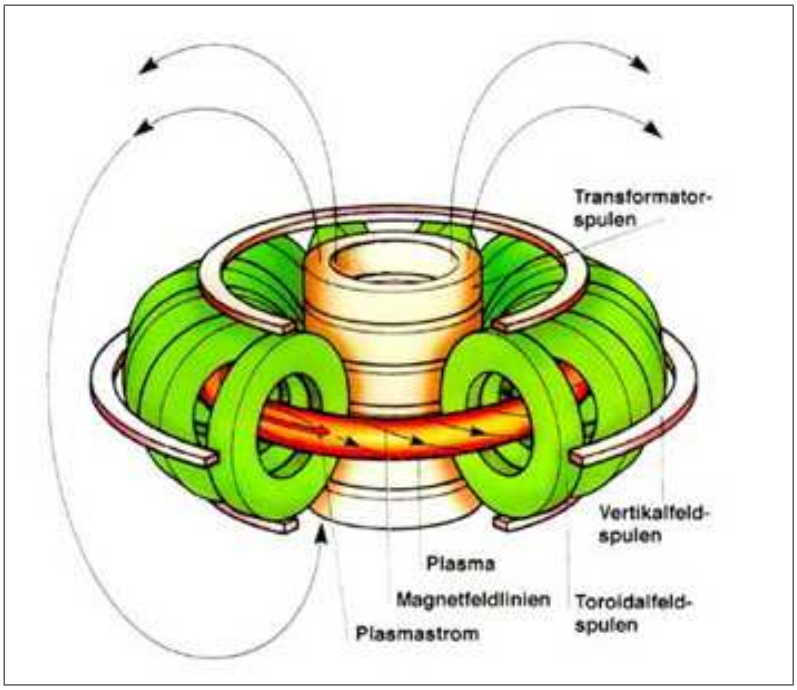


Abbildung 2.3: Das TOKAMAK-Prinzip (Quelle: www.euronuclear.org)

Trotz der starken Magnetfelder sind gelegentliche Instabilitäten des Plasmas nicht auszuschließen. Diese treten zum Teil auch im Inneren des Plasmas in Form von Strukturstörungen der Magnetfelder auf. Beim TOKAMAK werden diese sogenannten ”Moden”

in der Regel durch den Plasmastrom ausgelöst. Die Plasma-Instabilitäten reduzieren die freigesetzte Fusionsenergie und können das Plasma zerstören. Zudem können durch das kurzfristige Auftreffen sehr großer Teilchen- und Energiemengen auf eine kleine Fläche die Wände des Torus beschädigt werden. Die Gefahr einer Explosion oder einer unkontrollierten Kettenreaktion besteht jedoch nicht.

Um das Plasma auf die benötigte Temperatur zu bringen, werden beim TOKAMAK verschiedene Heizsysteme verwendet. Zunächst kann der im Inneren des Plasmas induzierte Strom als ohmsche Heizung verwendet werden. Eine weitere Heizmethode ist die Verwendung von Hochfrequenzheizungen. Dabei handelt es sich häufig um eine ”Elektron-Zyklotron-Resonanzheizung”, bei der die Elektronen mit Radiowellen erhitzt werden und ein sehr lokales Heizen möglich ist. Ein zusätzlicher Ansatz ist die Neutralteilcheninjektion, bei der elektrostatisch beschleunigte Ionen, die durch Umladung neutralisiert werden, in das Plasma eindringen. Abbildung 2.4 zeigt die verschiedenen Heizmethoden beim TOKAMAK.

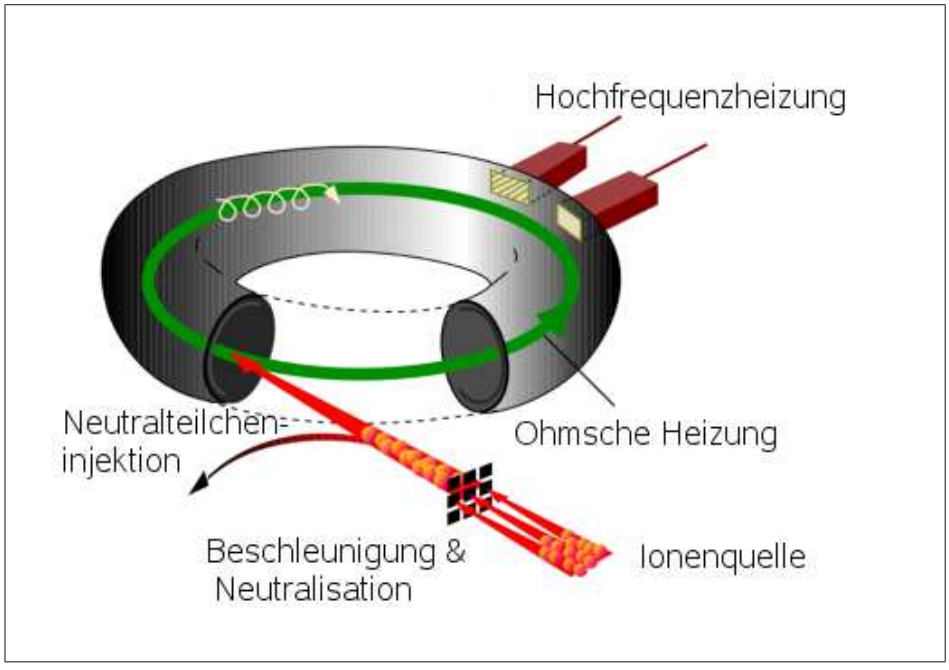


Abbildung 2.4: Heizmethoden beim TOKAMAK

Zwar sind verschiedene Heizmethoden bereits experimentell erprobt, jedoch steht die Übertragung für den Aufbau eines stabil laufenden Großkraftwerks noch bevor. Weitere Informationen zu Heizverfahren des Plasmas sind unter anderem in [WEB1] zu finden.

2.3 Eigenschaften und Ziele von TEXTOR

TEXTOR ist ein Experiment für technologieorientierte Forschung auf dem Gebiet der Plasma-Wand-Wechselwirkungen. Neben Teilchen- und Energieaustausch zwischen dem Plasma und der Wand können hier auch Maßnahmen zur Optimierung der Wand und des Plasmarandes untersucht werden, um Fragen der Matrialerosion und Disposition zu klären und die Freisetzung von Verunreinigungen und deren Einfluss auf den Plasmakern zu reduzieren. Für diese Zwecke besitzt der TEXTOR-Aufbau einige spezielle Eigenschaften und Komponenten:

- sehr gute Diagnostikverfahren für wandnahe Bereiche
- eine Vielzahl von Zugängen (Ports) für den Einsatz von Methoden zur Beeinflussung des Plasmarandes
- Einrichtungen zum Heizen des Gefäßes und der inneren Wand (Liner)
- Vorrichtungen zum Einbringen von Testmaterial mittels Limiterschleusen

Zusätzlich wurde TEXTOR im Laufe der Zeit mit einigen Erweiterungssystemen wie z.B. zusätzlichen Heizsystemen und einem Dynamisch Ergodischen Divertor (DED) versehen. Dieser bietet die Möglichkeit, Aspekte der Verbesserung des Plasmaeinschlusses sowie die Kontrolle der Wärme- und Teilchenabfuhr zu untersuchen. Weiterhin wird durch den DED mittels Erzeugung stochastischer Magnetfeldlinien am Plasmarand eine Reduktion der Wärmeflussdichte durch Verteilung der Wärme auf eine große Fläche erzielt und der Plasmatransport - insbesondere der Verunreinigungen - beeinflusst. Für die Wandkomponenten werden spezielle Materialien wie Graphit und Wolfram verwendet. Ein wichtiges Ziel ist die Vorhersage der Lebensdauer der Wandkomponenten und die Entwicklung entsprechender Werkstoffe. Dringliche Fragen, die in den nächsten Jahren unter anderem durch die Versuche an TEXTOR geklärt werden müssen, sind:

- die Identifizierung der Erosionsmechanismen und ihre Abhängigkeiten von Plasmabedingungen und Oberflächeneigenschaften für die verwendeten Werkstoffe
- die Untersuchung der potentiellen Tritiumrückhaltefähigkeiten der verschiedenen Oberflächenschichten

Zudem sollen auch die Verwendung des Dynamischen Ergodischen Divertors zur Reduzierung der Wandbelastung mit Hilfe von Störfeldern weiter untersucht sowie neue geeignete Diagnostiken entwickelt werden. Die Übersicht 2.1 stellt wichtige Kenndaten von TEXTOR dar. Weitere Informationen befinden sich in [WEB2].

Konfiguration	Tokamak
Plasmabegrenzung	Limiter / Dynamischer Ergodischer Divertor
Großer Plasmaradius	1,75 m
Mittlerer / kleiner Plasmaradius	0,47 m
Plasmavolumen	7 m^3
Anzahl der Hauptspulen	16
Magnetfeld	2,7 Tesla
Plasmastrom	$\leq$ 600 KA
Pulsdauer	$\leq$ 10 s
Installierte Heizleistung	9 MW
Heizmethoden	Neutralteilcheninjektion, Hochfrequenzheizverfahren für Elektronen und Ionen

Tabelle 2.1: Kenndaten von TEXTOR

2.4 Kapitelzusammenfassung

Bei der Kernfusion verschmelzen leichte Atomkerne unter Abgabe großer Energiemengen zu schwereren Kernen. Dies geschieht jedoch nur unter großem Druck und bei sehr hohen Temperaturen. In den letzten Jahrzehnten gab es viele Forschungsaktivitäten mit dem Ziel, die Kernfusion langfristig als Energiequelle nutzbar zu machen. Beim Großgerät TEXTOR im Forschungszentrum Jülich handelt es sich um ein Fusionsexperiment nach dem TOKAMAK-Prinzip, bei dem das durch starke Heizquellen in einem Torus erzeugte Fusionsplasma mittels starker überlagerter Magnetfelder von den Toruswänden ferngehalten wird. Die Untersuchung von Plasma-Wand-Wechselwirkungen, für die TEXTOR einige spezielle Komponenten besitzt, steht hierbei im Vordergrund.

Kapitel 3

Verarbeitung der Experimentdaten

In diesem Kapitel wird die Datenverarbeitung rund um das TEXTOR-Experiment betrachtet. Als erstes wird ein kurzer Überblick über die eingesetzten Messapparaturen und die Speicherung gemessener Daten auf einem zentralen Server gegeben, bevor auf die Verarbeitung der Messdaten und das Erstellen und Speichern physikalischer Signale eingegangen wird. Neben der Beschreibung der Speichersysteme (File Server CFS, Datenbank TPD) wird auch der Zugriff auf die Daten erklärt. Die aufgezeigten Probleme und Verbesserungsmöglichkeiten im Umgang mit der verwendeten Datenbank stellen die Motivation der Arbeit dar. Die konkreten Ziele der Arbeit werden am Ende des Kapitels aufgeführt.

3.1 Zentrale Speicherung von Messdaten

Zunächst wird das Vorgehen beim Erfassen von Messdaten unter Verwendung verschiedener Messapparaturen und das Sammeln der gemessenen Daten auf einem zentralen Fileserver dargestellt.

3.1.1 Aufzeichnung der Messdaten

Da es sich bei TEXTOR um einen TOKAMAK-Aufbau handelt, wird dieser wie in Kapitel 2.2.1 beschrieben nicht kontinuierlich, sondern gepulst betrieben. Als Puls oder auch Schuss werden die Vorgänge von der Plasmaerzeugung und dem Start des Fusionsprozesses bis zur vollständigen Beendigung des Plasmas bezeichnet. Schüsse beim TEXTOR-Experiment dauern ca. 10 Sekunden.
Es werden viele verschiedene Diagnostiken verwendet, die Informationen zu einem durchgeführten Schuss sammeln und dabei unterschiedliche Arten von Daten erzeugen. Vor ihrer Verwendung zur Datenerfassung müssen die Steuerungen und Datenaufzeichnungen der Diagnostiken in der Regel programmiert werden. Es werden sowohl spezielle, zum Teil selbst entwickelte Diagnostiken, als auch verfügbare Standarddiagnostiken für

die Messungen eingesetzt. Jedoch werden bei einem Schuss nicht immer alle Diagnostiken verwendet.

Die anfallenden Daten der Diagnostiken werden vorwiegend mittels Transientenrecorder über CAMAC (**C**omputer **A**utomated **M**easurement **A**nd **C**ontrol) erfasst. Dabei handelt es sich um ein Bussystem zur Datenerfassung und Experimentkontrolle in der Kern- und Teilchenphysik [WEB3]. Zur Steuerung und Datenaufzeichnung wird die CAMAC-Apparatur an einen PC angeschlossen, um Befehle zu empfangen und Daten zu transportieren. Im Laufe der Zeit hat sich die Steuersoftware zum Betreiben des CAMAC geändert. Aktuell wird die Steuerung mit der ”JDAQ”-Software durchgeführt (**J**ava **D**ata **Acq**uisition System). Diese speichert Daten unterschiedlicher Module. Eine genaue Beschreibung der JDAQ-Software befindet sich in [LIT04]. Neben Hardware zur Messdatenerfassung gibt es ebenfalls ein System zur Konfiguration der Maschinenkontrollparameter (”textorCC”, zuvor ”FBA”), welches unter anderem den Start der Aufzeichnungen koordiniert. Zudem kommen auch kamerabasierte Diagnostiken zum Einsatz. Die Anzahl der Diagnostiken wird sich im Laufe der Zeit durch Erweiterung vorhandener Module und den Einsatz neuer Technologien noch vergrößern.

Häufig werden den Signalnamen der Messapparaturen Aliasnamen zugewiesen, um eine bessere Übersicht zu erhalten. Dennoch kann die Filesystemstruktur für die Datenspeicherung komplex sein und auf Grund verschiedener Module einer Apparatur mehrere Unterverzeichnisse enthalten. Um die entsprechend formatierten Dateien zu erzeugen und auf dem zu einer Messapparatur gehörenden lokalen Server zu sichern bzw. diese anschließend auf einen zentralen Fileserver zu transportieren, werden Skripte und Programme (die bei der JDAQ-Software z.B. in JAVA implementiert sind) verwendet. Die Abläufe der Datenspeicherung werden dabei soweit wie möglich automatisiert.
Neben den eigentlichen Messdaten werden gewöhnlich auch zusätzliche Informationen zu den Parametern und Eigenschaften der Diagnostiken abgelegt. Zudem gibt es meist Beschreibungen (Metainformationen) zu den gemessenen Daten (z.B. Messbereich, Dimension etc.). Diese sind zur späteren Auswertung der Daten notwendig.

3.1.2 Transport der Rohdaten zum zentralen Fileserver CSF

Um einen zentralen und einheitlichen Zugriff auf die erfassten Daten verschiedener Diagnostiken bereitzustellen, wird ein sogenanntes ”Common Storage Facility” (CSF) betrieben, bei dem es sich um einen zentralen Fileserver handelt. Sollen bei einem Schuss aufgezeichnete Daten von dem lokalen Server einer Messapparatur zum CSF transferiert werden, geschieht dies nach folgendem Ablauf:

1. warten auf Startsignal (Trigger) und Abfragen der aktuellen Schussnummer

   (a) Erfassung und Speicherung der Daten auf dem lokalen Server

2. bevorstehende Datensendung beim CSF bekanntmachen
3. Daten übertragen
4. gesendete Daten überprüfen
5. Datensendung bestätigen

Die Struktur für die Speicherung der Daten (z.B. in Form von Unterverzeichnissen) ist wichtig, damit spätere Abfragen und Zugriffe auf die Daten möglichst einfach und übersichtlich erfolgen können. Sind die Übertragungsvorgänge und Speicherstrukturen einmal eingerichtet, laufen die Ladevorgänge größtenteils automatisch ab. Das Abrufen der aktuellen Schussnummer sowie das Ankündigen der Datensendung beim CSF kann durch das Aufrufen spezieller URLs aus den Programmen heraus erfolgen. Eine URL (Uniform Resource Locator) gibt den Ort und die Art des Zugriffs für eine Ressource im Internet an. Die Syntax und die Semantik bei der Verwendung von URLs ist durch das World Wide Web Consortium (W3C) standadisiert und in verschiedenen, vom W3C herausgegebenen Dokumenten (wie. z.B. [LIT05]) genau spezifiziert. Der eigentliche Datentransport wird in der Regel über FTP [1] oder SCP[2] durchgeführt. Durch den Aufruf der URL in (2.) werden entsprechende Verzeichnisse auf dem zentralen Fileserver erzeugt.

Das Prüfen und Bestätigen der gesendeten Daten kann ebenfalls über den Aufruf hierfür bereitgestellter URLs und das damit verbundene Ausführen entsprechender Programme (wie z.B. zur Prüfsummenbildung) geschehen. Erst nach der Bestätigung, dass die Daten vollständig und fehlerfrei zum CSF transportiert wurden, ist ein Zugriff auf die Daten durch Clientprogramme möglich. Abbildung 3.1 skizziert die zentrale Speicherung von Messdaten auf im CSF.

3.1.3 Eigenschaften des Common Storage Facility (CSF)

Das Common Storage Facility (CSF) ermöglicht durch die zentrale Speicherung einen einheitlichen Zugriff auf die Messdaten der Diagnostiken. Die zu verwaltende Datenmenge wächst dabei sehr rasch. Derzeit fallen pro Schuss ca. 3 Gigabyte neue Daten an. Mit zukünftigen Erweiterungen der Diagnostiken ist ein Anstieg auf 4 Gigabyte oder mehr möglich. Über die letzten Jahre ist ein exponentielles Wachstum der Gesamtmenge an Daten im CSF zu erkennen. Ende 2008 wurde daher ein Storage-System mit 50 Terrabyte Plattenplatz beschafft. Bei gleich bleibendem Wachstum ist jedoch in den nächsten Jahren eine weitere Vergrößerung der Speicherkapazität notwendig.

[1] FTP (File Transfer Protocoll): Netzwerkprotokoll zur Übertragung von Dateien in einem TCP/IP-Netzwerk

[2] SCP (Secure Copy): Protokoll zur verschlüsselten Datenübertragung

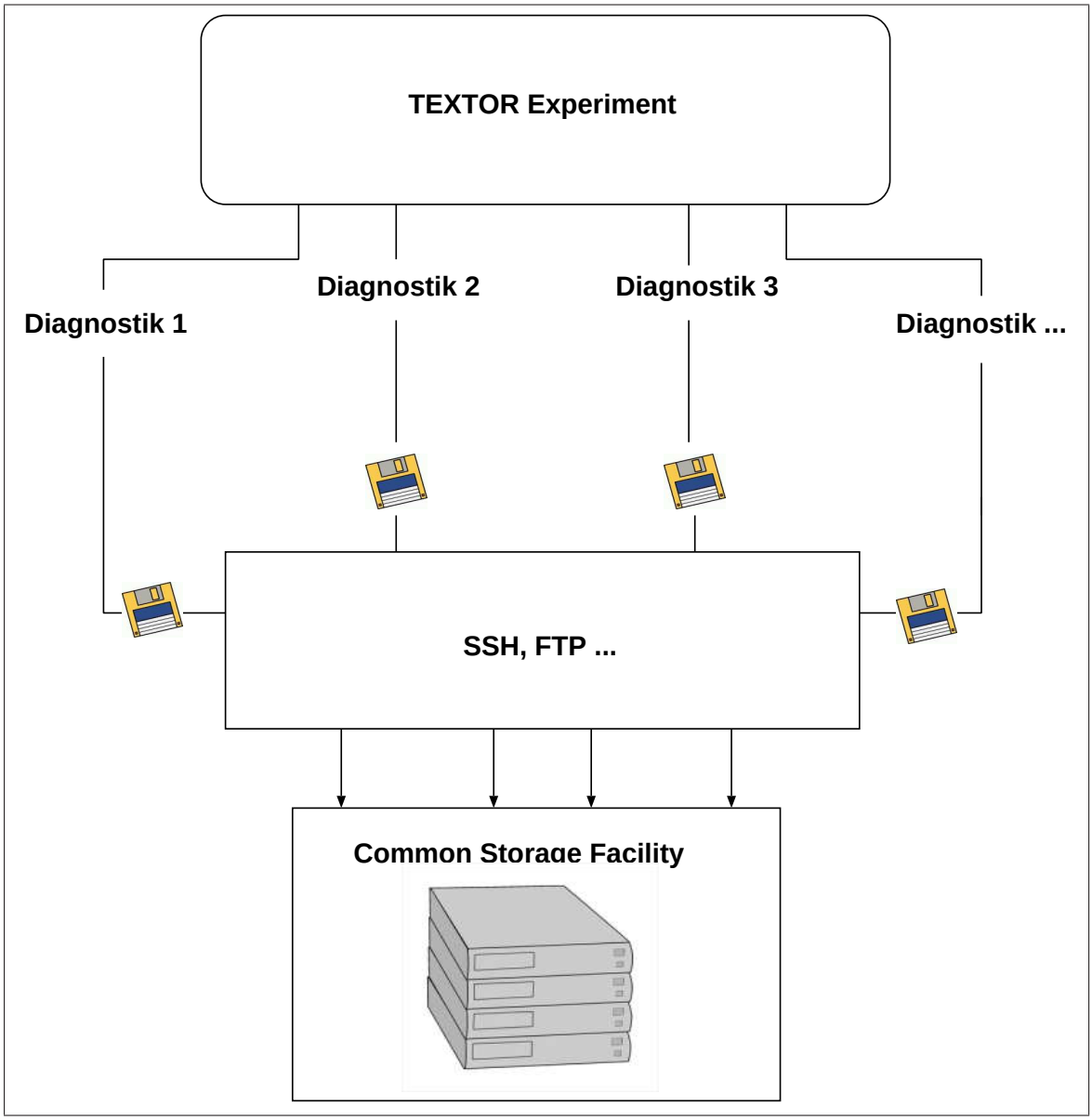


Abbildung 3.1: Zentrale Speicherung von Messdaten im Common Storage Facility

Um einen möglichst einheitlichen Abruf von Messdaten im CSF zur Verfügung stellen zu können, ist die Auswahl der Dateiformate derzeit beschränkt, kann jedoch bei Bedarf jederzeit erweitert werden. Neben ASCII-Formaten werden auch binäre Dateiformate eingesetzt. Zum Teil werden speziell auf eine Diagnostik angepasste Dateiformate für die Speicherung der Daten verwendet.

3.2 TEXTOR Physical Database

Neben einer allgemeinen Beschreibung der Rolle und der Eigenschaften der Datenbank wird im Folgenden auf das verwendete Datenbankmanagementsystem eingegangen und das Datenmodell bzw. die Speicherstrukturen beschrieben. Der Aufbau der verwendeten Tabellen und die Vorgänge beim Erzeugen neuer Signalversionen und dem Einfügen von Signaldaten in die Datenbank werden erklärt.

3.2.1 Eigenschaften und Ziele

Ziel der TEXTOR Physical Database (TPD) ist das Bereitstellen eines möglichst einfachen und zentralen Zugriffs auf die Daten physikalischer Signale des TEXTOR - Experiments. Auf Grund der Struktur der Signaldaten und der Anforderungen an die Abfrage bezüglich Auswahl und Suche von Daten, wird ein relationales Datenbanksystem für die TPD verwendet. Dies hat den Vorteil, dass die Abfrage der Signale über relativ einfache Befehle erfolgen kann, die sich zudem leicht in Weboberflächen integrieren lassen. Auf diese Weise können die Wissenschaftler relevante Daten ohne spezielle Datenbankkenntnisse im Web-Browser abfragen und durchsuchen.
In der TPD werden Daten gespeichert, die durch das Analysieren der von den Diagnostiken erstellten Rohdaten erzeugt werden. Diese abgeleiteten Signale besitzen in der Regel eine klare physikalische Bedeutung. Sie können jedoch meistens nicht direkt gemessen werden, sondern müssen aus den vorhandenen Messdaten berechnet werden. Die Berechnung der Signale erfolgt über verschiedene automatisch oder manuell gestartete Programme (s. hierzu auch Kapitel 3.2.4 auf Seite 20). Beispiele für physikalische Signale in der TPD sind etwa die ohmsche Heizleistung, der Plasmadruck oder die Elektronendichte.

3.2.2 Verwendetes Datenbanksystem

Als Datenbanksystem wird derzeit eine kommerzielle Datenbank der Firma Oracle® in der Version 10.2.0.1 verwendet. Bei der Software handelt es sich um ein objektrelationales Datenbankmanagementsystem (ORDBMS). Eine objektrelationale Datenbank erlaubt das Einbauen von Objekterweiterungen in die klassische relationale Datenbanktechnologie [LIT06, S. 2]. Unter einer Datenbank versteht man eine organisierte Zusammenstellung von Informationen [LIT07, Kap. 2, S. 11]. Das Datenbankmodell legt die Art der Beziehungen der Datensätze fest. Beim Relationalen Datenbankmodell werden die Informationen ausschließlich über Relationen (anschaulich: 2-dimensionale Tabellen) dargestellt. Dabei müssen folgende Grundanforderungen an die Tabellen erfüllt sein:

- es sind keine identischen Zeilen zulässig
- jede Zelle besitzt genau einen Wert
- die Spalten besitzen keine festgelegte Sortierung

Die Relationen können durch eine relationale Algebra spezifiziert werden, so dass eine mathematische Formalisierung möglich ist. Grundlagen für ein relationales Datenbankmodell wurden bereits 1970 von E.F. Codd in [LIT08] veröffentlicht. Das relationale Datenbankmodell setzte sich rasch gegenüber anderen Ansätzen, wie dem hierarchischen Modell oder der Netzwerkdatenbank, durch und ist derzeit das am häufigsten verwendete Modell.

Zur Verwaltung einer Datenbank wird ein Datenbankmanagementsystem (DBMS) benötigt. Hierzu zählt man die Programme zur Erzeugung, Verwaltung und Manipulation einer Datenbank. Das Datenbankmanagementsystem ermöglicht die Abstraktion der Datenorganisation, so dass die Datenspeicherung und der Zugriff durch die Anwender unabhängig vom Datenmodell geändert werden können. Beim DBMS handelt sich um eine komplexe Software, welche die Daten vewaltet und sie für Abfragen bereitstellt. Gleichzeitig überwacht und sichert das DBMS die Konsistenz der Daten und sorgt für Datenschutz und Datensicherheit [LIT09, S. 58].
Vorteile von relationalen Datenbanksystemen sind unter anderem die große Vielfalt an Modellierungen durch die Verwendung relationaler Operatoren und die standardisierte Abfrage über die Stuctured Query Language (SQL). Dabei handelt es sich um eine einfach zu benutzende Implementierung einer Abfragesprache mit der Anfragen an relationale Datenbanken gestellt werden können. Neben proprietären Varianten großer Datenbankhersteller existiert auch ein ANSI[3] -Standard. Oracle® bietet darüber hinaus mit PL/SQL eine prozedurale Erweiterung von SQL, mit der Programmeinheiten gespeichert und ausgeführt werden können. Somit können SQL-Befehle z. B. innerhalb von Schleifen und Funktionen ausgeführt werden. PL/SQL-Blöcke wie Prozeduren und Funktionen, die logisch zusammenhängen, können in sogenannten Packages zusammengefasst und bereitgestellt werden.
Oracle® verwendet bei der Verwaltung von Datenbankobjekten das Konzept von Schemas. Ein Schema ist einem User zugeordnet und kann als Sammlung all seiner Datenbankobjekte gesehen werden. Zu einem Schema gehören neben den Definitionen der Relationen (Tabellen) auch die Wertebereiche der Attribute (die in einer relationalen Datenbank als Spalten dargestellt werden).

[3]ANSI (American National Standards Institute): US-amerikanische Stelle zur Normung industrieller Verfahrensweisen

3.2.3 Speicherstruktur und Datenmodell

Wie bei einem relationealen Datenbankystem üblich, erfolgt die Speicherung der Daten in der TPD in Form von 2-dimensionalen Tabellen. Hierbei ist eine geschickte Aufteilung der Informationen auf die Tabellen nötig, um möglichst einfache und schnelle Abfragen zu ermöglichen.
Die in der TPD gespeicherten physikalischen Signale können derzeit in den Dimensionen 0, 1 oder 2 vorliegen. Je nach Dimension des Signals werden zur Speicherung 2 oder 3 Tabellen verwendet. Diese enthalten die Werte des entsprechenden Signals zu sämtlichen Schüssen, für die das Signal berechnet wurde, sowie eine Beschreibung des Signals. Die Verknüpfung der zu einem Signal gehörenden Tabellen erfolgt über die Schussnummern, die in allen Signaltabellen enthalten sind. Auf diese Weise können die Signalwerte stets einem Schuss zugeordnet werden. Eine Indizierung [4] der Signaltabellen an Hand der Schussnummern ermöglicht die Beschleunigung der Abfrage und der Suche von Schüssen und den zugehörigen Werten.

Für die Speicherung von 0-dimensionalen (Skalare) und 1-dimensionalen Signalen werden 2 Tabellen verwendet. Die erste Tabelle enthält eine Beschreibung des Signals und besitzt die Endung "`_description`". Hier werden Informationen wie die physikalische Einheit des berechneten Signals und der für die Signalerzeugung verantwortliche Mitarbeiter gespeichert. Die Tabelle 3.1 gibt einen Überblick über die vorkommenden Spalten einer solchen Beschreibungstabelle.
Die zweite Tabelle besitzt die Endung "`_bulk`" und enthält die eigentlichen Signalwerte zu den jeweiligen Schüssen und Versionen. Versionsnummern sind deshalb notwendig, weil Signale häufig mehrmals (z.B. mit unterschiedlichen Parametern) berechnet werden (vgl. Kapitel 3.4.1). Die 0-dimensionalen Signale besitzen nur einen Wert pro Schussnummer und Version. Ein Beispiel hierfür ist die Schussdauer. Die Bulktabelle eines 0-dimensionalen Signals ist demnach wie in Tabelle 3.2 dargestellt aufgebaut.

[4] Ein Index ist ein vom Oracle®-Server automatisch verwaltetes Schemaobjekt in der Datenbank, das den Abruf von Zeilen durch Verwendung von Zeigern beschleunigt.

Spaltenname	**Beschreibung**
SHTID	Schussnummer
DIMENSIONS	Anzahl Dimensionen (0, 1 oder 2)
LENGTH_TOTAL	Anzahl der Zeilen
LENGTH_0	Länge der zugehörigen Bulk-Tabelle
START_0	Startzeit
STOP_0	Endzeit
INC_0	ggf. feste Schrittweite zwischen Start- und Endzeit
UNIT_0	Einheit der X-Achse
LABEL_0	Bezeichnung der X-Achse
LENGTH_0	ggf. Länge der Abszissen-Tabelle
START_1	ggf. Startzeit
STOP_1	ggf. Endzeit
INC_1	ggf. feste Schrittweite zwischen Start- und Endzeit
UNIT_1	ggf. Einheit der zweiten Messachse
LABEL_1	ggf. Bezeichnung der zweiten Messachse
UNIT	Einheit der Messwerte
LABEL	Bezeichnung der Messwerte
PRC_LVL	Prozess-Level
PRC_PROC	Programm, das zur Signalerzeugung verwendet wurde
QUALITY	ggf. Informationen zur Qualität der Messwerte
RESP	verantwortlicher Mitarbeiter
DESCRIPTION	Beschreibung des Signals
CREA_DATE	Datum der Signalerzeugung
VERSION	Versionsnummer
DEPENDENCIES	Abhängigkeiten von anderen Signalen

Tabelle 3.1: Aufbau einer Tabelle zur Beschreibung eines Signals

Spaltenname	**Beschreibung**
SHTID	Schussnummer
ORDINATE	Messwert
VERSION	Versionsnummer

Tabelle 3.2: Aufbau der ”`_bulk`”-Tabelle für ein 0-dimensionales Signal

Für 1-dimensionale Signale, wie z.B. die Heizleistung ”PTOT”, sind mehrere Wertepaare pro Schuss und Version vorhanden. Bei der Abszisse handelt es sich in der Regel um die Zeitachse. Die Ordinate spiegelt dann die jeweiligen Werte zu den bestimmten Zeitpunkten wieder. Bulktabellen für 1-dimensionale Tabellen sind wie in Tabelle 3.3 dargestellt aufgebaut.

Spaltenname	**Beschreibung**
SHTID	Schussnummer
ABSCISSA	Wert auf der zweiten Messachse
ORDINATE	Messwert
VERSION	Versionsnummer

Tabelle 3.3: Aufbau der ”`_bulk`”-Tabelle für ein 1-dimensionales Signal

Die 2-dimensionalen Signale hängen von einer weiteren Dimension, wie z.B. einem Radius ab. Ein Beispiel hierfür ist das Elektronentemperatur-Profil. Bei 2-dimensionalen Signalen ist eine dritte Tabelle vorhanden. Neben der Beschreibungstabelle, die wie bei den Skalaren und 1-dimensionalen Signalen aufgebaut ist, werden hier zwei Tabellen für die Speicherung der Werte verwendet. Die Tabelle mit der Endung ”`_abscissa_0`” enthält die Werte für die zweite Messachse.

Spaltenname	**Beschreibung**
SHTID	Schussnummer
ABSCISSA_0	Wert auf der Messachse
VERSION	Versionsnummer

Tabelle 3.4: Aufbau einer ”`_abscissa_0`”-Tabelle

Die Bulktabelle enthält wie bei den Skalaren und 1-dimensionalen Signalen die eigentlichen Signalwerte und besitzt die Endung ”`_bulk`”. Allerdings ist sie auf Grund der Verwendung zweier Messachsen anders aufgebaut (s. Tabelle 3.5). Die zweite Messachse bei 2-dimensionalen Signalen ist formal zunächst auf hundert unterschiedliche Werte beschränkt (Spalten VALxxx), ließe sich bei Bedarf jedoch durch Einfügen neuer Spalten erweitern. Sind weniger als hundert Messpunkte auf der zweiten Messachse vorhanden, werden die entsprechenden Spalten mit Nullwerten [5] aufgefüllt.

[5] NULL-Wert: nicht verfügbarer, nicht zugewiesener, unbekannter oder nicht anwendbarer Wert; nicht zu verwechseln mit der Zahl 0 oder einem Leerzeichen

Spaltenname	Beschreibung
SHTID	Schussnummer
ABSCISSA_1	Wert auf der ersten Messachse
VERSION	Versionsnummer
VAL000	Messwert an Position 0 der zweiten Messachse
VAL001	Messwert an Position 1 der zweiten Messachse
⋮	⋮
VAL099	Messwert an Position 99 der zweiten Messachse

Tabelle 3.5: Aufbau der "`_bulk`"-Tabelle für ein 2-dimensionales Signal

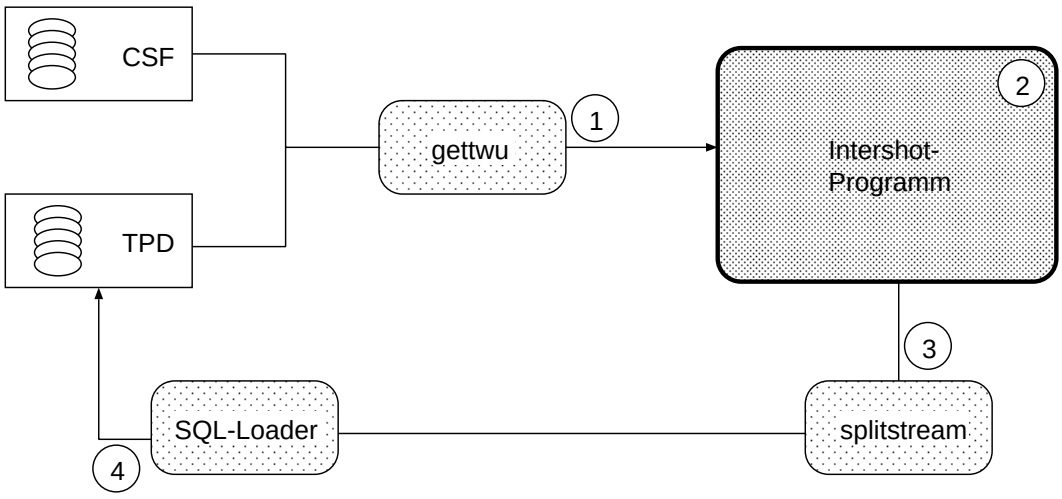
3.2.4 Berechnung von Signalen

Nach jedem Schuss werden automatisch verschiedene Programme zur Berechnung der physikalischen Signale gestartet. Diese werten die von den Diagnostiken aufgezeichneten Daten aus und fügen die berechneten Signale in die TPD ein. Dieser Vorgang wird als "Automatic Intershot Processing" bezeichnet. Hierzu wird ein Webserver verwendet, über CGI[6]-Skripte die notwendigen Berechnungsprogramme aufruft. Die Sammlung der automatisch ausgeführten Programme trägt den Namen "Chain1". Damit die Programme stetig weiterentwickelt werden und von mehreren Entwicklern bearbeitet werden können, wird ein CVS-System[7] eingesetzt. Neben den eigentlichen Programmen werden auch verschiedene Skripte und Makefiles eingesetzt, um die Abläufe der Signalberechnung zu automatisieren.
Nach einem Schuss wird die aktuelle Schussnummer an die Chain1 übergeben und es werden nacheinander die Berechnungsprogramme ausgeführt, wobei ggf. vorhandene Abhängigkeiten bei der Reihenfolge der Ausführungen berücksichtigt werden. Die einzelnen Programme arbeiten nach dem Prinzip von Eingabe, Verarbeitung und Ausgabe (EVA). Zunächst werden die benötigten Eingabedaten eingelesen, womit dann die entsprechenden Berechnungen ausgeführt werden. Die Ergebnisse werden anschließend in Form von Signalen in die TPD eingefügt. Abbildung 3.2 skizziert die Ausführung eines Intershot-Programms.

Zu den einzelnen Programmen gehören neben den Quellcodes in der Regel weitere Dateien, in denen z.B. die für das jeweilige Programm benötigten Eingabedaten aufgeführt oder Berechnungsparameter festgelegt werden. Die Programme können in verschiedenen Programmiersprachen erstellt werden. Es muss jedoch ein entsprechendes Makefile zur

[6]Common Gateway Interface: Schnittstelle, über die ein Webserver mit weiterer Software interagieren kann

[7]CVS (Concurrent Versions System): Software zur Versionsverwaltung von Dateien



1. Über das Programm 'gettwu' werden die zur Berechnung benötigten Signale aus der TPD bzw. Rohdaten aus dem CSF gelesen.

2. Die Berechnungen werden über das Intershot-Programm ausgeführt.

3. Die Ergebnisse werden an das Skript 'splitstream' zur Aufteilung in Dateien übergeben.

4. Die neu berechneten Signale werden aus den Dateien mit Hilfe des SQL-Loaders in die TPD geladen.

Abbildung 3.2: Ablauf der Ausführung eines Intershot-Programms

Übersetzung bereitgestellt werden. Wenn ein neues Programm erstellt und ausreichend getestet wurde, kann es in die Chain1 integriert werden. Hierzu müssen die notwendigen Skripte, wie z.B. das für den Aufruf der Chain1 - Programme zuständige Makefile angepasst werden. Zusätzlich muss die TPD für das Einfügen des neuen Signals durch Anlegen weiterer Tabellen vorbereitet werden, damit das neue Signal dort eingefügt werden kann. Das Laden der Signale in die TPD erfolgt über den "SQL-Loader". Dabei handelt es sich um ein von Oracle® bereitgestelltes Software-Tool, mit dem sich Daten automatisch aus Dateien in die Datenbank laden lassen. Diese Methode ist besonders gut für große Mengen strukturierter Daten geeignet, wie sie bei den Intershot-Programmen anfallen.
Neben den Programmen aus der Chain1, die automatisch nach jedem Schuss gestartet werden, gibt es weitere Auswertungsprogramme, die jedoch manuell angestoßen und ggf. nur bei bestimmten Schüssen gestartet werden. Auch diese Programme können unter Verwendung von Shell-Skripten und des SQL-Loader Signale in die TPD einfügen. Zudem können die in der Chain1 vorhandenen Programme ebenfalls manuell gestartet werden.

3.3 Einheitlicher Datenzugriff über TEC Web Umbrella

Da der Zugriff auf die gemessenen und berechneten Daten eine wichtige Rolle für die Auswertung des Experiments spielt, werden hier die Anforderungen und die Umsetzung bei der Abfrage von Daten beschrieben.

3.3.1 Anforderungen an die Datenabfrage

Bei jedem neuen Schuss fällt eine große Menge an Daten an, die von verschiedenen Systemen erzeugt und an unterschiedlichen Speicherorten (z.B. CSF und TPD) abgelegt werden. Zudem liegen die Daten in unterschiedlichen Speicherformaten vor. Dabei kann es sich um einzelne Werte, konkrete Messdaten oder berechnete Signale handeln. Bei der Verwaltung all dieser Daten gilt es, folgenden Anforderungen gerecht zu werden:

- Den am Experiment beteiligten Wissenschaftlern und Technikern soll ein zentraler Zugang zu sämtlichen Daten bereitgestellt werden.
- Der Zugriff auf die verschiedenen Daten sollte möglichst intuitiv und einheitlich sein.
- Die zu Grunde liegenden Speicherformen sollen abstrahiert werden.
- Parallele Anfragen müssen möglich sein und effizient bearbeitet werden (Verarbeitung konkurrierender Zugriffe).
- Eine Selektion von Daten (Abfrage von Teilmengen) ist bereitzustellen.
- Die Darstellung abgefragter Daten soll möglichst einheitlich sein.

Allgemein sollen die Daten portabel (also möglichst systemunabhängig) und leicht zu verarbeiten sein.

3.3.2 Das Abfragesystem TEC Web Umbrella

Um die oben genannten Anforderungen bei den Datenzugriffen zu erfüllen, wurde das Abfragesystem ”TEC Web Umbrella” (TWU) entwickelt. Wie der Name schon andeutet, wird hiermit ein zentraler webbasierter Zugriff auf die verschiedenen Datenquellen ermöglicht. Abbildung 3.3 veranschaulicht den Einsatz des TWU-Protokolls für den Zugriff auf die Daten von CSF und TPD.

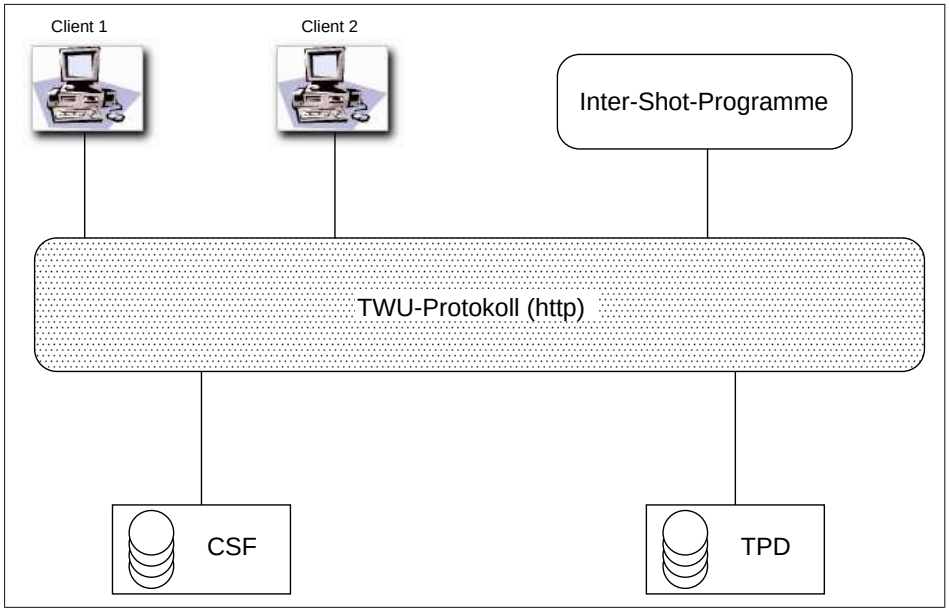


Abbildung 3.3: Verwendung des TEC Web Umbrella für den einheitlichen Zugriff auf die Experimentdaten

Die Grundidee des TWU ist, die allgmein als ”Signale” bezeichneten Daten unabhängig von der Speicherform als Verzeichnisse und Dateien auf einem Webserver darzustellen. ”Everything is (made look like) a web-page” ([LIT10, S. 5, Z. 8]). Der TWU-Zugriff basiert auf einem zentralen Webserver und dem HTTP-Protokoll. Dabei handelt es sich um ein vom World Wide Web Consortium (W3C) standardisiertes Protokoll auf Anwendungsebene, das neben der Übertragung von Hypertext (wie z.B. Webseiten) auch für die Übertragung anderer Daten und Informationen verwendet werden kann. Die zur Zeit aktuelle Protokollversion HTTP/1.1 wurde bereits 1999 in [LIT11] spezifiert, wird jedoch derzeit überarbeitet.[8] Das HTTP-Protokoll bietet unter anderem folgende Vorteile:

- es ermöglicht eine einfache Nutzung über einen Webbrowser
- gut getestete und frei verfügbare Software (z.B. Webserver, Browser) für verschiedene Plattformen ist vorhanden
- das HTTP-Protokoll ist gut geeignet, um verschiedene Datentypen zu transportieren
- effiziente Verwaltung konkurrierender Zugriffe ist bereits im Webserver integriert

Nicht alle Arten von Daten sind für die Darstellung im Browser geeignet. Daher werden für die Auswertungen weitere Programme, wie z.B. Matlab, verwendet. Die Abfrage

[8] siehe hierzu auch http://www.w3.org/Protocols/#News

und der Transport der Daten kann aber auch hierbei über das HTTP-Protokoll erfolgen. Neben dem Browserzugriff werden zusätzlich Bibliotheken für gängige Programmiersprachen wie C und Fortran bereitgestellt, um einen Zugriff auf das TEC Web Umbrella zu ermöglichen.
Die abzufragenden Signale werden über entsprechende URLs angesprochen. Dabei gibt es für die URLs konkrete Vorgaben, um die Abbildung auf die abzufragenden Signale zu regeln. Dies ermöglicht eine Automatisierung der Abfrage bei den auswertenden Programmen. Zum Beispiel lassen sich die Werte der Elektronendichte im zuletzt durchgeführten Schuss über folgende URL abrufen:

```
http://ipptwu.ipp.kfa-juelich.de/textor/all/0/tpd/ne
```

Wie an der Beispiel-URL zu sehen ist, werden die für die Abfrage benötigten Schuss- und Signal-informationen nicht wie bei URLs üblich in From von ”get-Parametern”, die durch ”?” getrennt an den Pfad der URL angehängt werden, übergeben. Stattdessen werden diese Informationen als Pfadangaben in die URL eingebaut. Signale und Schussnummern werden dabei wie Verzeichnisse in der URL angegeben. Das Format (die Pfadangabe) der URL ist genau festgelegt:

```
http://<host>/<exp>/<grouping>/<shotnumber>/<store>/ \
              <subsys>/<opt>/<signalname>
```

- `<host>`
  Der Hostname des Webservers (derzeit: ipptwu.ipp.kfa-juelich.de)
- `<exp>`
  Name des Experiments, aus dem die Daten erzeugt worden. Neben TEXTOR können hier auch die Daten weiterer Fusionsexperimente eingebunden werden.
- `<grouping>`
  Ermöglicht die Gruppierung von Schussnummern, z.B. an Hand eines Zeitintervals, um die Schussmenge zu reduzieren. Die Gruppe ”all”, in der alle Schüsse enthalten sind, muss bei jedem Experiment zur Verfügung stehen.
- `<shotnumber>`
  Die Schussnummer.
- `<store>`
  Gibt die Datenquelle an. Hier z.B. die TPD oder JDAQ
- `<subsys>`
  Eine Untergruppe, zu der eine Diagnostik gehört.
- `<opt>`
  Optionale weitere Eingruppierung des Signals

- `<signalname>`
  Name des abzufragenden Signals.

Der Pfad der URL wird von einer serverseitigen Software interpretiert, die daraus eine entsprechende Abfrage an die jeweilige Datenquelle (z.B. CSF oder TPD) generiert und die angeforderten Daten an den Browser zurückschickt.
Das Format für die Ausgabe der Signaldaten ist ebenfalls genau spezifiziert, sowohl für die Signalbeschreibung als auch für die eigentlichen Werte. Die Kodierung der Ausgabe von Signalen bei Abfragen des TWU erfolgt als 7-bit ASCII-Text.[9] Dies hat den Vorteil, dass die Ausgabe von allen Programmiersprachen verarbeitet werden kann, die URL-Inhalte lesen können.

Die Signale können bei der Abfrage bereits durch das TWU aufbereitet werden. Möglich ist etwa die Skalierung von Signalwerten, die Einschränkung auf einen Teilbereich eines Signals oder die Durchführung einfacher Berechnungen (z.B. Multiplikation der Signalwerte mit einem Faktor). Diese zusätzlichen Optionen werden in Form von Parametern an die URL angehängt. In der folgenden URL wird durch Angabe der Parameter ”start”, ”step” und ”total” ein bestimmter Teilbereich der Signaldaten abgfragt. Zudem wird eine Schrittweite zur weiteren Einschränkung der Daten vorgegeben.

```
http://ipptwu/textor/all/95010/tpd/ne/V1/bulk?start=120&step=5&total=10
```

Durch die Angabe eines zusätzlichen Parameters (”terse”) kann festgelegt werden, ob die Ausgabe HTML[10]-Anweisungen (z.B. zur Verwendung von Links) enthalten soll oder nicht. Die Nutzung von HTML-Anweisungen kann bei der Ausgabe z.B. für die Darstellung im Browser nützlich sein, bei der Verwendung von Auswertungsprogrammen wie Matlab jedoch stören.

Es ist ebenfalls möglich, künstliche Signale zu erzeugen, die nicht gemessen wurden, sondern über Funktionen im TWU bereitgestellt werden. So lässt sich z. B. eine Näherung der Kreiszahl π über folgende URL als Signal abfragen:

```
http://ipptwu/textor/all/constants/C=3.1415/U=Hz/almostPI
```

Neben den Signalen, bei denen es sich im Wesentlichen um die Messdaten auf dem CSF und die berechneten Signale der TPD handelt, stellt die Einstiegsseite des TWU auch Links für den Zugriff auf die technischen Logbücher des Experiments sowie News und Dokumentation rund um TEXTOR bereit. Diese Informationen werden auf eigenen, von Java-Servlets generierten Webseiten dargestellt.

[9] ASCII (America Standard Code for Information Interchange): standadisierte Kodierung von Zeichensätzen

[10] HTML (Hypetext Markup Language): im World Wide Web verwendete und vom Browser interpretierte Sprache zur Strukturierung von Texten, Bildern und Referenzierungen

3.4 Probleme bei der Datenverarbeitung und mögliche Verbesserungen

Im Hinblick auf Probleme und Verbesserungsmöglichkeiten bei der zuvor beschriebenen Verarbeitung der Experimentdaten stehen hier die automatische Aktualisierung von Signalversionen sowie die Performance der Datenbankabfragen im Mittelpunkt. Diese stellen zugleich die Motivation der Arbeit dar.

3.4.1 Versionen abhängiger Signale in der TPD

Signale in der TPD können in unterschiedlichen Versionen zu einem Schuss vorliegen. Für die Auswertung der physikalischen Signale und den Erhalt aussagekräftiger Schlussfolgerungen ist es notwendig, dass voneinander abhängige Signale in einer aktuellen Version vorliegen. Um dies zu gewährleisten, müssen nach jeder Neuberechnung eines Signals auch alle von diesem Signal abhängigen Signale in der TPD neu berechnet werden. Da die Neuberechnung eines Signals jedoch für gewöhnlich manuell angestoßen wird, werden die Signalabhängigkeiten in der Regel nicht geprüft. Das heißt, dass die abhängigen Signale nicht automatisch neu berechnet werden. Wird die aktuelle Version eines Signals abgefragt, kann diese also eventuell auf Eingangsdaten basieren, die in der Zwischenzeit aktualisiert wurden und daher unter Umständen veraltet sind.
Weiterhin werden derzeit keine Informationen darüber gespeichert, aus welchem Grund ein Signal neu berechnet wurde. Gerade im Hinblick auf die Abhängigkeiten zwischen den Signalen wäre eine solche Information jedoch hilfreich.

3.4.2 Performance bei TPD-Abfragen

Abfragen der TPD werden an mehreren Stellen ausgeführt, zum einen bei der Berechnung von Signalen (sowohl bei der automatischen Ausführung der Chain1 als auch bei manuellen Berechnungen) und zum anderen bei der Auswertung von Signalen. Die Geschwindigkeit für die Signalabfragen aus der TPD spielt daher eine große Rolle. Problematisch ist hierbei die Größe der Signaltabellen. Pro Schuss werden dort große Anzahlen neuer Zeilen (bei manchen Signalen bis zu mehreren tausend) eingefügt. Die Signaltabellen derjenigen Signale, die bei jedem Schuss berechnet werden, können daher mehrere Millionen Zeilen beinhalten. Dies stellt ein Problem für die Handhabung und die Performance dar. Trotz der Indizierung der Schussnummern kommt es häufig zu langsamen Abfragen.
Eine weitere Schwierigkeit ist die Variation der ausgeführten Abfragen. Zum Teil werden komplette Signaldaten zu einem oder mehreren Schüssen abgefragt, es können jedoch auch Bereichseinschränkungen und die Suche nach Schüssen mit bestimmten Merkmalen auftreten. Neben der Indizierung der Schussnummern sind also weitere Maßnahmen notwendig, um die verschiedenen Typen von Abfragen der TDP zu beschleunigen.

3.5 Ziele der Arbeit

Aus den beschriebenen Verbesserungsmöglichkeiten bei der Datenverarbeitung ergeben sich folgende Ziele, die im Rahmen der Arbeit umgesetzt werden sollen.

3.5.1 Automatisierung der Neuberechnung von Signalen

Um die Konsistenz der Versionen abhängiger Signale in der TPD zu gewährleisten, soll eine automatische Neuberechnung abhängiger Signale implementiert werden. Dabei sollen Signale (ohne manuellen Eingriff) neu berechnet werden, sobald eine neue Version eines der für die Berechnung verwendeten Signale erstellt wurde. Die Mitarbeiter brauchen folglich bei der Berechnung neuer Signalversionen keine Abhängigkeiten der Signale zu berücksichtigen. Betroffen sind lediglich die manuellen Aufrufe der Berechnungsprogramme, da bei der Ausführung der Chain1 (also der Erzeugung der ersten Signalversion zu einem Schuss) Signalabhängigkeiten ohnehin automatisch berücksichtigt werden. Die Durchführung der automatisch angestoßenen Neuberechnungen abhängiger Signale soll unter Verwendung der vorhandenen Intershot-Programme geschehen. Zusätzlich soll nachvollziehbar sein, warum das entsprechende Signal neu berechnet worden ist.

3.5.2 Verwaltung zusätzlicher Metadaten

Beschreibungen und Metainformationen zu den einzelnen Schüssen und Signalen sind bereits in der TPD enthalten. So ist z.B. vermerkt, mit welchem Ziel und unter welchen Voraussetzungen ein bestimmter Schuss durchgeführt wurde. Im Rahmen der Arbeit soll eine Erweiterung der Informationen auf die Signalberechnungen und deren Versionen durchgeführt werden. Dadurch soll nachvollziehbar sein, warum die jeweilige Version eines Signals berechnet wurde. Die zusätzlichen Metainformationen sollen dabei in die TPD integriert werden. Für die komfortable Abfrage dieser Informationen soll ein Web-Zugriff bereitgestellt werden, der sich in die vorhandenen Webseiten integrieren lässt. Nach Möglichkeit sollen die zusätzlichen Informationen automatisch erstellt und in die TPD eingefügt werden.

3.5.3 Performanceverbesserung der Datenbankabfragen

Abfragen in der TPD dauern unter Umständen sehr lange und erschweren ein produktives Arbeiten mit den vorhandenen Signaldaten. Das verwendete Oracle®-Datenbanksystem stellt verschiedene Techniken zur Optimierung von Abfragen und deren Performance bereit. Es soll untersucht werden, wie durch den Einsatz von gezielten Maßnahmen die Abfragezeiten verbessert werden können.

Da das Hauptproblem die Größe der Tabellen ist, sollen die Signaltabellen durch ein ”Partitioning”[11] aufgeteilt und die Auswirkungen des Partionierens auf die Abfragezeiten untersucht werden. Weiterhin soll geprüft werden, ob der Einsatz zusätzlicher Indizes einen positiven Einfluss auf die Geschwindigkeit von Abfragen hat. Ebenfalls soll der mögliche Einsatz von Materialized Views[12] untersucht werden. Mit Hilfe von Zeitmessungen beispielhafter Anfragen sollen konkrete Aussagen hergeleitet und die möglichen Verbesserungen im Produktionssystem umgesetzt werden.

3.6 Kapitelzusammenfassung

Beim TEXTOR-Experiment werden viele verschiedene Diagnostiken eingesetzt, die unterschiedliche Arten von Daten liefern. Die Messdaten werden zentral auf einem Fileserver (Common Storage Facility) gespeichert. Aus diesen Rohdaten werden anschließend mit verschiedenen Programmen physikalische Signale berechnet und in Tabellenform in der TEXTOR Physical Database (TPD), einer objektrelationalen Datenbank, abgelegt. Einen einheitlichen Zugriff, sowohl auf die Signale in der TPD als auch auf die Rohdaten des zentralen Fileservers, in Form von Webseiten bietet das Tec Web Umbrella (TWU) unter Verwendung des HTTP-Protokolls.
Die automatische Aktualisierung abhängiger Signale in der TPD beim Erstellen einer neuen Signalversion ist bisher nicht gewährleistet und soll im Rahmen der Arbeit implementiert werden. Zudem sollen zusätzliche Metainformationen zu den einzelnen Signalversionen erzeugt und über das TWU verfügbar gemacht werden. Darüber hinaus ist eine mögliche Optimierung der Performance von Signalabfragen aus der TPD zu untersuchen.

[11] Partitioning: Aufteilen von Tabellen und/ oder Indizes in kleinere Einheiten

[12] Materialized Views: Im Gegensatz zu herkömmlichen Views werden für die Beschleunigung von Zugriffen hierbei die Daten physikalisch gespeichert

Kapitel 4

Automatische Neuberechnung von Signalen der TPD

In diesem Kapitel wird die automatische Berechnung neuer Signalversionen beschrieben. Dabei werden die gegebenen Voraussetzungen erläutert und anschließend detailliert die Umsetzung dargestellt.

4.1 Anforderungen und Abläufe bei der Neuberechnung

Um abhängige Signale in der TPD aktuell zu halten, müssen automatische Neuberechnungen der abhängigen Signale durchgeführt werden. Hierzu muss ein Mechanismus gefunden werden, der automatisch reagiert, sobald eine neue Version eines Signals in der TPD erstellt wird. Die Erstellung der ersten Version eines Signals, die durch das automatische Ausführen der Chain1 - Programme nach jedem Schuss erzeugt wird, soll davon ausgeschlossen werden, da die Chain1 vorhandene Abhängigkeiten unter den Signalen bereits berücksichtigt. In Abbildung 4.1 werden die Abläufe der automatischen Neuberechnung skizziert.
Wird eine neue Signalversion (mit Versionsnummer > 1) erstellt, muss geprüft werden, ob es Signale gibt, die von diesem gerade neu berechneten Signal abhängen. Durch die Auswertung der vorhandenen Metainformationen zu den Signalen und ihren Abhängigkeiten muss also eine Liste aller neu zu berechnender Signale erstellt werden. Anschließend müssen die Intershot-Programme zur Berechnung der neuen Signalversionen mit den entsprechenden Parametern aufgerufen werden. Nach der erfolgreichen Neuberechung der abhängigen Signale und dem Einfügen der neuen Signalversionen in die TPD sollen Metainformationen zu den durchgeführten Berechnungen und der neuen Signalversion erzeugt und ebenfalls in die TPD eingefügt werden.

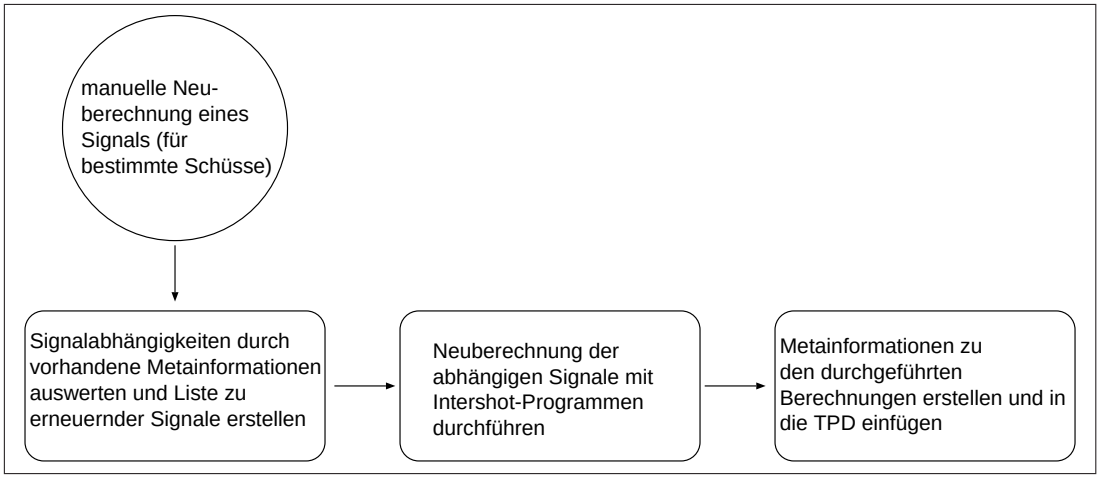


Abbildung 4.1: Ablauf der automatischen Neuberechnung von Signalen

Da nur direkte Abhängigkeiten in den Beschreibungstabellen der TPD vermerkt sind, wird das Auflösen der Abhängigkeiten erschwert. Als Beispiel seien die in Abbildung 4.2 dargestellten Abhängigkeiten für einen Schuss gegeben.

Abbildung 4.2: Indirekte Signalabhängigkeiten in der TPD

Im vorliegenden Fall müssten bei einer manuellen Neuberechnung von Signal 1 sowohl Signal 2, welches direkt von Signal 1 abhängt, als auch Signal 3, welches nur indirekt von Signal 1 abhängt, sowie ggf. weitere abhängige Signale automatisch neu berechnet werden. Signal 1 wird jedoch in der Beschreibungstabelle von Signal 3 nicht als Abhängigkeit geführt, sondern lediglich Signal 2.
Ein konkretes Beispiel für einen solchen Fall stellt der Schuss 94988 für die Abhängigkeit des Signals ”PTOT” von dem Signal ”NELFJC” dar. Wie in der schematischen Darstellung der TPD-Abhängigkeiten in Abbildung 4.3 zu erkennen ist, hängt das Signal ”PTOT” nur indirekt über die Signale ”PECRH” und ”NE” vom Signal ”NELFJC” ab. Die Signale ”POH”, ”PNBI”, ”TE” und ”NELFJC” besitzen für diesen Schuss keine Signalabhängigkeiten in der TPD.

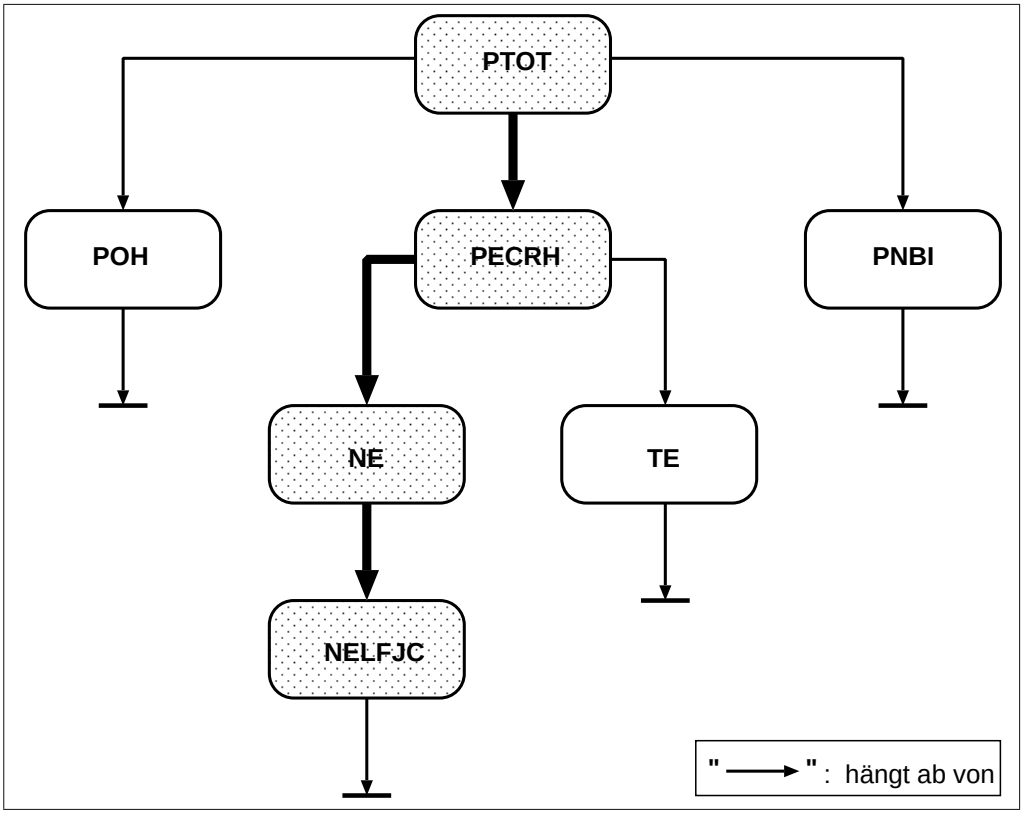


Abbildung 4.3: Beispiel für indirekte Signalabhängigkeiten in der TPD

Um für diesen Schuss eine Abhängigkeit zwischen dem Signal ”PTOT” und dem Signal ”NELFJC” festzustellen, müssen also die Beschreibungstabellen der drei Signale ”PTOT”, ”PECRH” und ”NE” ausgewertet werden. Die Tabellen 4.1, 4.2 und 4.3 zeigen die entsprechenden Informationen aus den Beschreibungstabellen zu den aufgezeigten Abhängigkeiten.

shtid	**dependencies**
...	...
94988	tpd/poh tpd/pnbi star/icrhi star/icrhii **tpd/pecrh**
...	...

Tabelle 4.1: Abhängigkeiten des Signals PTOT für Schuss 94988

shtid	dependencies
...	...
94988	**tpd/ne** tpd/te jdaq/IVD/IBT2P-star fom/ecrh/100kHz_ADCs/Signals/Forward_Power Elevation Rotation star/ip
...	...

Tabelle 4.2: Abhängigkeiten des Signals PECRH für Schuss 94988

shtid	dependencies
...	...
94988	star/nel144 star/nel157 star/nel165 star/nel175 star/nel185 star/nel195 star/nel205 star/nel215 **tpd/nelfjc**
...	...

Tabelle 4.3: Abhängigkeiten des Signals NE für Schuss 94988

Der erstellte Automatismus muss vor dem Einsatz im Produktionssystem unter Berücksichtigung eventuell auftretender Sonderfälle untersucht werden.

4.2 Implementierung

Zunächst werden die einzelnen Techniken eingeführt, die im Rahmen der Implementierung von Bedeutung sind. Dabei spielen besonders der Mechanismus von Triggern und die Verwendung von PL/SQL-Prozeduren eine wichtige Rolle. Die einzelnen Abläufe und Vorgehensweisen wie die Suche nach Abhängigkeiten und das Starten notwendiger Neuberechnungen werden beschrieben und dabei das jeweilige Vorgehen begründet.

4.2.1 Verwendung von Triggern und Stored Procedures

Für das automatische Durchführen von Aktionen auf Grund bestimmter Ereignisse stellt das verwendete Oracle®-Datenbanksystem den Mechanismus der Trigger bereit. Dabei handelt es sich um in der Datenbank gespeicherte PL/SQL-Code-Objekte, die bei bestimmten Ereignissen automatisch ausgeführt werden. Verschiedene Aktionen können als auslösende Ereignisse dienen, beispielsweise das Einfügen in eine Tabelle, die Anmeldung eines Benutzers bei der Datenbank oder der Versuch eines Benutzers, eine Tabelle zu löschen.

Trigger in der Oracle®-Datenbank, die auf DML-Ereignisse (Data Manipulating Language) wie das Einfügen, Aktualisieren oder Löschen von Daten reagieren, besitzen die in Tabelle 4.4 dargestellten Attribute.

Attribut	**Mögliche Werte**	**Beschreibung**
Trigger Zeitpunkt	BEFORE, AFTER, INSTEAD OF	Gibt an, wann ein Trigger ausgelöst wird
Trigger Ereignis	INSERT, UPDATE, DELETE	Operation, die den Trigger auslöst
Trigger Typ	STATEMENT, ROW	Gibt an, wie oft der Trigger ausgeführt wird: einmal pro Statement oder für jede Zeile
Trigger Aktion	PL/SQL Block	Aktion, die beim Auslösen des Triggers ausgeführt wird

Tabelle 4.4: Attribute von DML-Triggern in der Oracle®-Datenbank

Der Zeitpunkt für das Auslösen eines Triggers ist dann wichtig, wenn der Trigger einen Einfluss auf die zu bearbeitenden Daten hat. So kann z.B. ein Trigger, der vor dem Einfügen einer neuen Tabellenzeilge ausgelöst wird, die einzufügenden Daten ändern oder ergänzen. Bei der automatischen Signalberechnung in der TPD reicht es jedoch aus, den entsprechenden Trigger *nach* dem Einfügen einer neuen Zeile in eine der Beschreibungstabellen auszulösen, da die einzufügenden Zeilen unverändert übernommen werden sollen. Die für die automatische Signalberechnung verwendeten Trigger besitzen daher folgende Eigenschaften:

- Trigger Zeitpunkt: AFTER
- Trigger Ereignis: INSERT
- Trigger Typ: ROW

Da nur für diejenigen der eingefügte Zeilen, in denen die Versionsnummer größer als 1 ist, eine Prüfung der Abhängigkeiten durchgeführt werden soll, müssen die Trigger zusätzlich eine entsprechende WHEN-Klausel enthalten. Trigger können andere PL/SQL-Prozeduren und -Funktionen aufrufen. ”Es empfiehlt sich, den Code des Triggers sehr kurz zu halten und alles, für das längerer Code erforderlich ist, in einem separaten Package zusammenzufassen[LIT12, Kap.8, S. 19, Z. 6ff].”

Das Prüfen der Signalabhängigkeiten und das Starten von Berechnungen wird daher durch den Aufruf der PL/SQL-Prozedur ”solveDependencies” realisiert. Dabei handelt es sich um eine ”Stored Procedure”, die in übersetzter Form auf dem Oracle®-Server abgelegt wird und wie andere Datenbankobjekte (z.B. Tabellen oder Trigger) automatisch verwaltet wird. Die Verwendung von Stored Procedures ist von Vorteil für die Datensicherheit der Tabellendaten, da Stored Procedures mit den Privilegien des Eigentümers der Prozedur ausgeführt werden. Zudem wird durch das Entfallen der Übersetzungszeit bei der Ausführung in der Regel eine gute Performance erzielt. Bei der Verwendung im Client-Server-Betrieb kann das Zusammenfassen mehrerer SQL-Anweisungen zu einer Stored Procedure zusätzlich den Netzwerkverkehr reduzieren und somit zu einem Zeitgewinn führen [LIT13, S. 230f].

Da alle Signale in der TPD berücksichtigt werden sollen, muss für jede Beschreibungstabelle ein Trigger erstellt werden. Der PL/SQL-Code in Abbildung 4.4 erstellt einen entsprechenden Trigger für die Tabelle ”NE_DESCRIPTION”. Die Trigger für die übrigen Beschreibungstabellen der Signale in der TPD wurden auf die gleiche Weise erzeugt. Als Parameter an die Prozedur ”solveDependencies” werden der Signalname, die Schussnummer, die Versionsnummer und das Erstellungsdatum der neu eingefügten Signalversion übergeben.

```
CREATE OR REPLACE TRIGGER ne_desc_trg
AFTER INSERT ON ne_description FOR EACH ROW
WHEN (NEW.version > 1)
BEGIN
   solveDependencies ('NE', :NEW.shtid, :NEW.version,
                      :NEW.crea_date);
END;
```

Abbildung 4.4: PL/SQL-Statement zum Erzeugen des Triggers NE_DESC_TRG

4.2.2 Vermeidung von Kollisionen

Durch die Verwendung von Triggern wird sichergestellt, dass bei jedem Einfügen einer neuen Signalversion in eine der Beschreibungstabellen die Prozedur ”solveDependencies” zum Auflösen von Abhängigkeiten und für das automatische Starten von Neuberechnungen aufgerufen wird. Dabei kann es vorkommen, dass auf Grund einer durchgeführten Neuberechnung und des damit verbundenen Einfügens einer neuen Signalversion in die TPD der nächste Trigger ausgelöst wird, bevor der erste Aufruf der Prozedur ”solveDependencies” abgeschlossen ist.

Der mehrfache Aufruf der PL/SQL-Prozedur führt jedoch zu Problemen. So kann die von der Prozedur erzeugte Logdatei nicht chronologisch geordnet geschrieben werden. Neben dem Verlust der Übersichtlichtkeit und Nachvollziehbarkeit durchgeführter Aktionen besteht die Gefahr, dass sich die verschiedenen Prozesse, die gleichzeitig schreibend auf die Logdatei zugreifen, gegenseitig behindern oder blockieren und die Prozedur sowie das ursprüngliche Einfügen in die Beschreibungstabelle nicht erfolgreich ausgeführt werden können.
Da der Zugriff auf eine Beschreibungstabelle solange blockiert ist, bis der Trigger (also in diesem Fall die im Trigger aufgerufene Prozedur) erfolgreich ausgeführt worden ist, führt der Zugriff auf die entsprechende Tabelle beim mehrfachen Aufruf der Prozedur ”solveDependencies” zu dem in Abbildung 4.5 dargestellten Oracle®-internen Fehler ”ORA-04091”.

04091, 00000, ”table %s.%s is mutating, trigger/function may not see it”
***Cause:** A trigger (or a user defined plsql function that is referenced in this statement) attempted to look at (or modify) a table that was in the middle of being modified by the statement which fired it.
***Action:** Rewrite the trigger (or function) so it does not read that table.

Abbildung 4.5: Fehler bei mehrfachem Tabellenzugriff

Ein Beispiel für das Auftreten einer solchen Kollision ist die in Abbildung 4.6 dargestellte Neuberechnung des Signals ”NE” für den Schuss 95010. Wie in der Abbildung zu sehen ist, überschneiden sich die durch die Trigger ausgelösten Prozeduraufrufe.
Das mehrfache (gleichzeitige) Aufrufen der Prozedur ”solveDependencies” muss also unbedingt vermieden werden. Eine erneute Ausführung der Prozedur darf erst erfolgen, wenn der vorherige Aufruf erfolgreich abgeschlossen wurde.
Um dies zu gewährleisten, wird eine Lock-Datei verwendet, in der vermerkt wird, ob sich die Prozedur aktuell bereits in der Ausführung befindet. Diese Lock-Datei wird am Anfang der Prozedur gelesen und an Hand deren Inhalt geprüft, ob die Prozedur bereits ausgeführt wird. Ist dies der Fall (Lock-Datei enthält ”running”), wird die Prozedur beendet. Andernfalls wird die Lock-Datei editiert (Inhalt von ”ready” auf ”running” ändern) und die Auflösung der Abhängigkeiten durchgeführt. Nach der Ausführung wird die Lockdatei wieder angepasst (Inhalt von ”running” auf ”ready” ändern), um einen erneuten Aufruf der Prozedur zu ermöglichen.

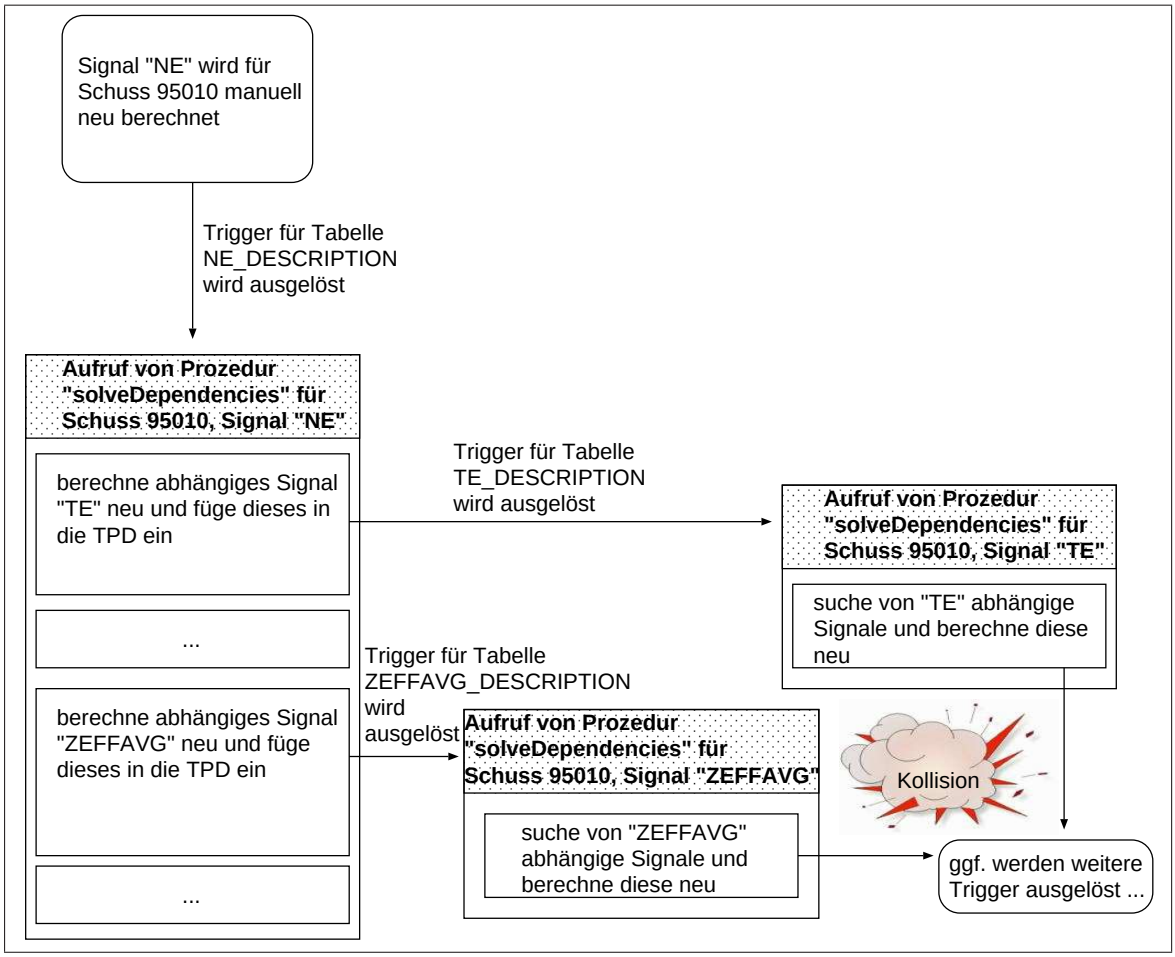


Abbildung 4.6: Überschneidung mehrerer Prozeduraufrufe

Durch den in Abbildung 4.7 skizzierten Mechanismus wird sichergestellt, dass sich die Prozedur nur einmal in der Ausführung befinden kann. Allerdings geht dadurch ein wesentlicher Vorteil der Verwendung von Triggern verloren. Die in Kapitel 4.1 beschriebenen indirekten Abhängigkeiten werden nun nicht mehr automatisch durch das aufeinanderfolgende Auslösen der Trigger aufgelöst. Durch die angestoßene Prozedur müssen also sämtliche Abhängigkeiten (direkte und indirekte Abhängigkeiten) aufgelöst werden.

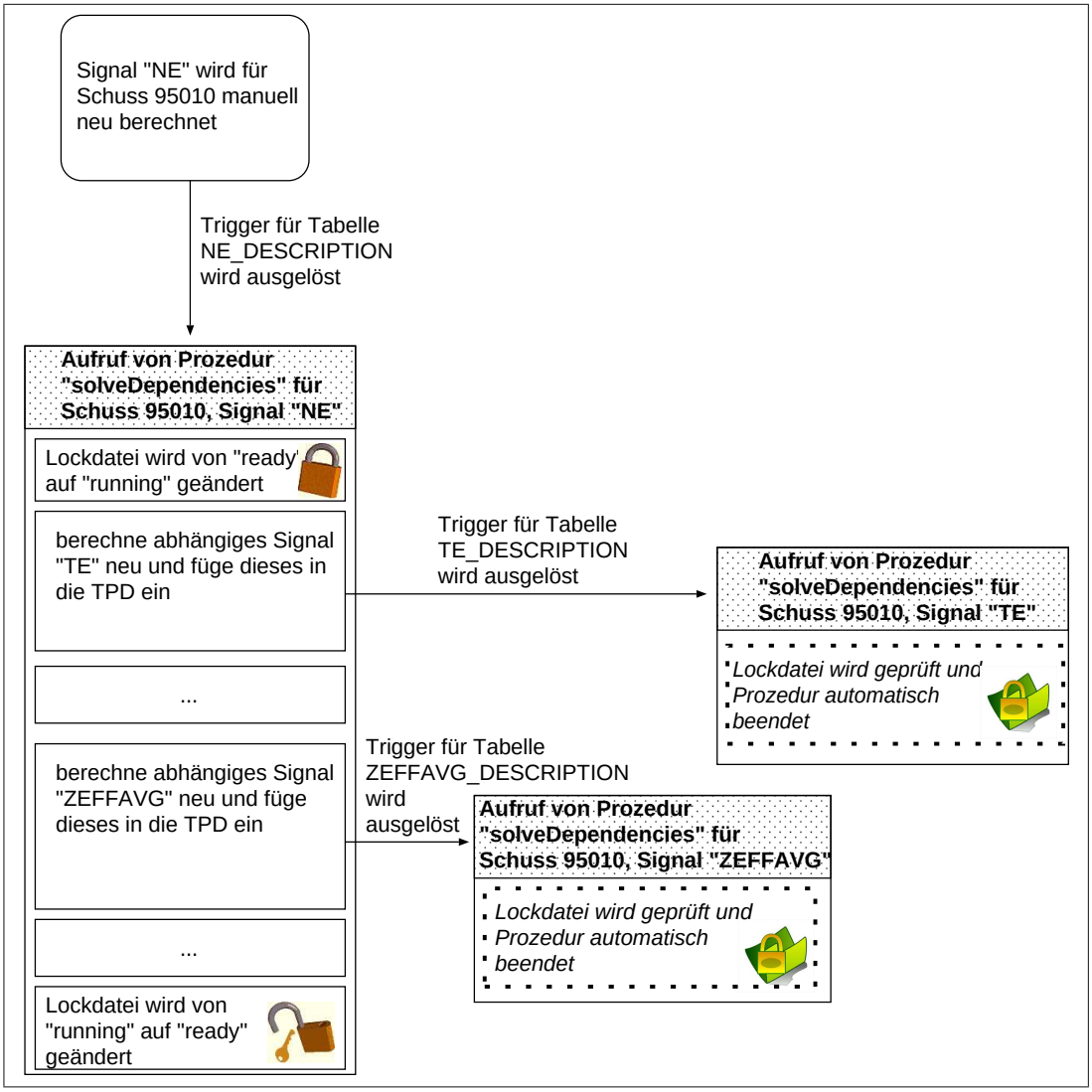


Abbildung 4.7: Vermeidung von Kollisionen durch Verwendung einer Lock-Datei

Darüber hinaus muss ggf. auch bei der manuellen Neuberechnung durch den Anwender der Status der automatischen Neuberechnungen berücksichtigt werden. Das heißt, eine manuelle Neuberechnung darf nur dann angestoßen werden, wenn keine automatischen Neuberechnungen durchgeführt werden und sich die Prozedur ”solveDependencies” aktuell nicht in der Ausführung befindet.

4.2.3 Feststellen von Abhängigkeiten

Zu einem manuell neu berechneten Signal müssen alle für den jeweiligen Schuss abhängigen Signale in der TPD gefunden werden, um diese ebenfalls neu zu berechnen. Hierzu wird zunächst die Tabelle ”TPD_STORE” abgefragt. Diese enthält zu allen Schüssen die jeweils verfügbaren TPD-Signale. Die Tabelle 4.5 zeigt einen Ausschnitt dieser Informationen.

shtid	**signal_list**
...	...
92881	HCNI10,HCNI20,HCNI30,HCNI40,HCNI60,HCNI70,HCNI80, HCNI90,NE,nela,POH,PNBI1,PNBI2,
92882	HCNI10,HCNI20,HCNI30,HCNI40,HCNI60,HCNI70,HCNI80, HCNI90,NE,nela,POH,PNBI1,PNBI2,
92883	HCNI10,HCNI20,HCNI30,HCNI40,HCNI60,HCNI70,HCNI80, HCNI90,NE,nela,POH,PNBI1,PNBI2,
92884	HCNI10,HCNI20,HCNI30,HCNI40,HCNI60,HCNI70,HCNI80, HCNI90,NE,nela,POH,PNBI1,PNBI2,
92885	nela,PNBI1,PNBI2,
92886	
92887	PNBI1,PNBI2,
92888	PNBI1,PNBI2,
92889	PNBI1,PNBI2,
...	...

Tabelle 4.5: Auszug aus der Tabelle TPD_STORE

Wie zu erkennen ist, kann die Signalliste für bestimmte Schüsse auch leer sein. Dieser Sonderfall muss bei der späteren Verarbeitung der Liste berücksichtigt werden. Die einzelnen Signale lassen sich über die von PL/SQL in Form von Funktionen bereitgestellten Stringoperationen, wie z.B. INSTR (zur Textsuche innerhalb einer Zeichenkette) und SUBSTR (zum Ausschneiden eines Teilstücks aus einer Zeichenkette) extrahieren.

Da nicht für jeden Schuss alle Signale berechnet werden, lässt sich durch die Abfrage der Tabelle ”TPD_STORE” die Anzahl möglicher abhängiger Signale auf diejenigen Signale beschränken, die für den bestimmten Schuss auch tatsächlich berechnet wurden. Dadurch kann die Anzahl auf Abhängigkeiten zu prüfender Signale ggf. stark reduziert werden. Um zu prüfen, ob für die möglichen abhängigen Signale tatsächlich eine Abhängigkeit besteht, muss jeweils die Spalte ”DEPENDENCIES” aus der Beschreibungstabelle des Signals ausgewertet werden. Hierbei sind nur die Informationen zur aktuellen Version und der vorgegebenen Schussnummer von Interesse. Für eine entsprechende Abfrage kann der in Abbildung 4.8 dargestellte SQL-Code (mit der jeweiligen Schussnummer und dem entsprechenden Tabellennamen) verwendet werden.

```
SELECT shtid, version, dependencies
FROM ne_description
WHERE shtid = 95689 AND version = (
   SELECT max(version) FROM ne_description
   WHERE shtid = 95689
);
```

Abbildung 4.8: SQL-Statement zur Abfrage von Signalabhängigkeiten

Die so erhaltene Liste von Abhängigkeiten kann dann erneut mit Hilfe der in PL/SQL zur Verfügung stehenden Funktionen zur Bearbeitung von Zeichenketten ausgewertet werden. Allerdings sind die Signale in einem TWU-Format gespeichert, das heißt, die Herkunft des Signals wird mit angegeben. Für die automatische Neuberechnung sind lediglich die Signale aus der TPD, die den Pfad ”tpd/” enthalten, relevant. Der in Tabelle 4.6 dargestellte Auszug aus der Tabelle NE_DESCRIPTION zeigt die Liste von Abhänigkeiten für das Signal NE (das Profil der Elektronendichte) in der Version 1 für den Schuss 95689. Relevant für das Auflösen von Abhängigkeiten wäre in diesem Fall nur der Eintrag ”tpd/nelfjc”, da es sich bei den übrigen Einträgen nicht um TPD-Signale handelt.

shtid	**version**	**dependencies**
...	...	...
95689	1	star/nel144 star/nel157 star/nel165 star/nel175 star/nel185 star/nel195 star/nel205 star/nel215 **tpd/nelfjc**
...	...	...

Tabelle 4.6: Abhängigkeiten des Signals ”NE” im Schuss 95689

Die einzelnen Signale aus den in der Spalte ”DEPENDENCIES” aufgeführten Abhängigkeiten werden mit dem Signal, durch welches die Prozedur angestoßen wurde, verglichen. Wird das Signal in der Liste der Abhängigkeiten gefunden, muss das zur Beschreibungstabelle zugehörige Signal für den vorgegebenen Schuss neu berechnet werden.
Als Beispiel wird in Abbildung 4.9 das Vorgehen bei der Suche nach Abhängigkeiten für eine Neuberechnung des Signals ”PECRH” im Schuss 95064 skizziert. Die Abfrage der Tabelle ”TPD_STORE” liefert für diesen Schuss die Signale ”POH”, ”PNBI1”, ”PNBI2” und ”PTOT”. Für diese wird jeweils die Spalte ”DEPENDENCIES” aus der Beschreibungstabelle für den vorgegebenen Schuss ausgewertet. Wie zu erkennen ist, besitzt nur das Signal ”PTOT” Abhängigkeiten in der TPD. Weil darin auch das Signal ”PECRH” enthalten ist, muss daher eine Neuberechnung des Signals ”PTOT” angestoßen werden.

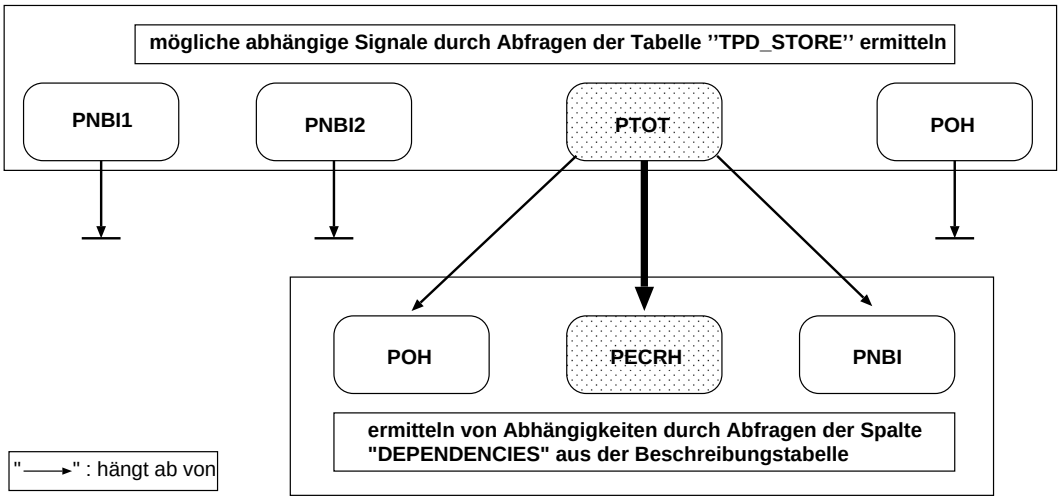


Abbildung 4.9: Beispiel für eine einfache Suche nach abhängigen Signalen

Durch das zuvor beschriebene Vorgehen werden jedoch lediglich Signale mit einer direkten Abhängigkeit zu dem manuell neu berechneten Signal gefunden. Um sämtliche (auch die indirekten) Abhängigkeiten aufzulösen, ist ein rekursives Vorgehen notwendig. Dieses wird dadurch erschwert, dass direkte und indirekte Abhängigkeiten in Kombination auftreten können. Dies ist z.B. für den Schuss 95010 und die Signale ”NE”, ”TE”, und ”ZEFFAVG” der Fall. Wie in den Auszügen aus den Beschreibungstabellen (in den Tabellen 4.7 und 4.8) zu erkennen ist, hängt das Signal ”ZEFFAVG” im Schuss 95010 sowohl vom Signal ”NE” als auch vom Signal ”TE” ab, welches wiederum selbst vom Signal ”NE” abhängig ist. ”ZEFFAVG” besitzt für diesen Schuss also sowohl eine direkte als auch eine indirekte Abhängigkeit zum Signal ”NE”. Abbildung 4.10 verdeutlicht die Problematik.

shtid	dependencies
...	...
95010	/star/bt /star/curr **/tpd/ne** /jdaq/mwz/ec107 /jdaq/mwz/ec109 /jdaq/mwz/ec112 /jdaq/mwz/ec115 /jdaq/mwz/ec120 /jdaq/mwz/ec125 /jdaq/mwz/ec130 /jdaq/mwz/ ec135 /jdaq/mwz/ec141 /jdaq/mwz/ec145 /jdaq/mwz/ec
...	...

Tabelle 4.7: Abhängigkeiten des Signals ”TE” im Schuss 95010

shtid	dependencies
...	...
95010	**tpd/ne tpd/te** brems/icamii/brems_ch10
...	...

Tabelle 4.8: Abhängigkeiten des Signals ”ZEFFAVG” im Schuss 95010

Um mehrfahre Neuberechnungen zu vermeiden und den Rechenaufwand zu reduzieren, darf eine Neuberechnung für ein Signal erst starten, wenn *ALLE* dem Signal zu Grunde liegenden Signale bereits neu berechnet wurden. Nach einer manuell angestoßenen Neuberechnung des Signals ”NE” für den Schuss 95010 müsste also zuerst das Signal ”TE” und anschließend das Signal ”ZEFFAVG” neu berechnet werden, da andernfalls das Signal ”ZEFFAVG” auf Grund der Abhängigkeit von ”TE” unter Umständen zweimal neu berechnet würde.

Stellt man die Signalabhängigkeiten wie in Abbildung 4.10 zu sehen als Baum mit dem abhängigen Signal als Wurzel dar, so darf das abhängige Signal erst dann neu berechnet werden, wenn alle Signale auf dem Weg von der Wurzel (hier ”ZEFFAVG”) bis zu dem Signal, zu dem eine Abhängigkeit besteht (hier ”NE”), bereits berechnet wurden, im vorliegenden Fall also Signal das Signal ”TE”.

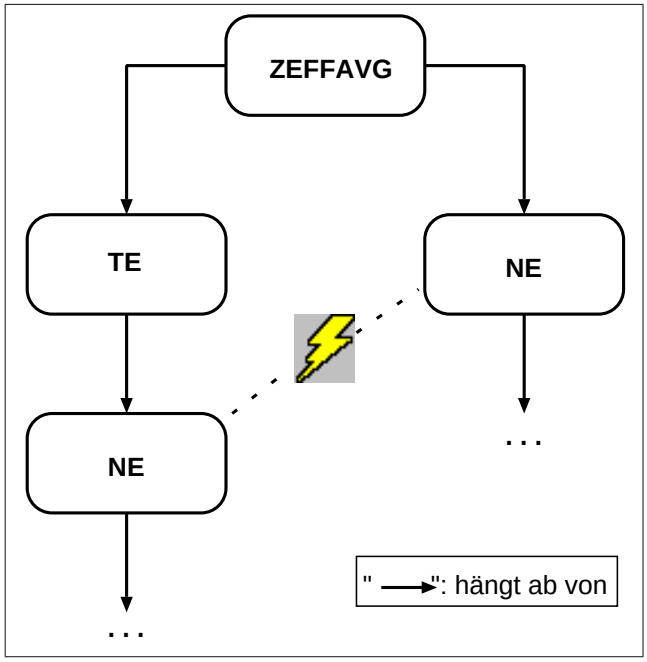


Abbildung 4.10: Beispiel für das Auftreten einer direkten und indirekten Abhängigkeit

Damit das Durchführen von Neuberechnungen in der richtigen Reihenfolge gewährleistet werden kann, wird daher für jedes zur Neuberechnung in Frage kommende Signal nicht nur geprüft, ob eine Abhängigkeit besteht, sondern auch ein Abhängigkeitslevel bestimmt. Dies entspricht in der Baum-Darstellung der Anzahl Äste, die von der Wurzel bis zu dem entsprechenden Signal durchlaufen werden müssen. Gibt es mehrere mögliche Wege, so muss der längste Weg gewählt werden, damit die richtige Reihenfolge der Neuberechnungen eingehalten werden kann.
Im vorigen Beispiel muss also dem Signal ”TE” das Level ”1” (direkte Abhängigkeit) und dem Signal ”ZEFFAVG” das Level ”2” (indirekte Abhängigkeit über Signal ”TE”) zugewiesen werden. Bei den durchzuführenden Neuberechnungen der abhängigen Signale werden die Signale mit dem niedrigsten Abhängigkeitslevel zuerst berechnet, diejenigen mit höherem Abhängigkeitslevel anschließend. Besitzen zwei Signale das gleiche Abhängigkeitslevel, ist ihre Berechnungsreihenfolge frei wählbar, da unter diesen beiden Signalen auf Grund der Vorgehensweise bei der Bestimmung der Abhängigkeitslevel keine Abhängigkeit bestehen kann.
Das Bestimmen des Abhängigkeitslevels für die in Frage kommenden Signale erfolgt durch den Aufruf einer rekursiven Funktion ”getDependencyLevel”. An diese Funktion werden beim Aufruf zwei Signalnamen, eine Schussnummer, eine Liste der zu diesem Schuss berechneten TPD-Signale und ein Abhängigkeitslevel übergeben. Für das erste

übergebene Signal soll ein Abhängigkeitslevel bestimmt werden. Das zweite übergebene Signal ist das, zu dem ggf. eine Abhängigkeit besteht, beim ersten Aufruf der Funktion also genau das Signal, zu dessen Beschreibungstabelle der entsprechende Trigger ausgelöst wurde. Als Abhängigkeitslevel wird zunächst ”0” ($\simeq$ keine Abhängigkeit) übergeben. In Abbildung 4.11 ist die Deklaration der verwendeten rekursiven Funktion dargestellt.

```
FUNCTION getDependencyLevel (
   v_signal_1 IN VARCHAR2,
   v_signal_2 IN VARCHAR2,
   v_shot_number IN NUMBER,
   t_signal_list IN string_array_type,
   v_level IN NUMBER )
RETURN NUMBER
```

Abbildung 4.11: Deklaration der rekursiven Funktion ”getDependencyLevel”

Die Länge der Liste möglicher abhängiger Signale (`t_signal_list`) kann als obere Grenze für das Abhängigkeitslevel verwendet werden, um eine Endlosrekursion zu vermeiden, die durch fehlerhafte Einträge in der TPD (z.B. gegenseitige Abhängigkeit zweier Signale) auftreten könnte. Ist das übergebene Abhängigkeitslevel kleiner oder gleich der Länge der Liste, wird zunächst das aktuelle Abhängigkeitslevel auf ”0” gesetzt.
Danach werden in der Funktion alle relevanten Abhängigkeiten von Signal 1 für den übergebenen Schuss ermittelt. Ist diese Liste leer (z.B. wenn das entsprechende Signal lediglich aus den gemessenen Rohdaten erzeugt wurde und keine Abhängigkeiten in der TPD besitzt), so gibt die Funktion den Wert ”0” als Abhängigkeitslevel zurück.
Andernfalls werden die Signalnamen der durch das Abfragen der Spalte ”DEPENDENCIES” ermittelten relevanten Abhängigkeiten der Reihe nach in einer Schleife betrachtet. Handelt es sich bei dem Signalnamen um Signal 2, so wird das aktuelle Abhängigkeitslevel mit dem an die Funktion übergebenen Level verglichen. Ist das aktuelle Abhängigkeitslevel kleiner oder gleich dem übergebenen Level, so erhält es den um 1 erhöhten Wert des übergebenen Levels.
Enstpricht der betrachtete Signalname nicht Signal 2, so wird das Abhängigkeitslevel über einen erneuten Aufruf der Funktion ermittelt. Dabei wird als Signal 1 der gerade betrachtete Signalname übergeben und als Level das um 1 erhöhte Eingangslevel. Ist das so ermittelte Abhängigkeitslevel größer als das aktuelle Abhängigkeitslevel, so wird es übernommen. Wurden alle relevanten Abhängigkeiten durchlaufen, wird das aktuelle Level zurückgegeben. Abbildung 4.12 skizziert das Vorgehen innerhalb der Funktion ”getDependencyLevel”.

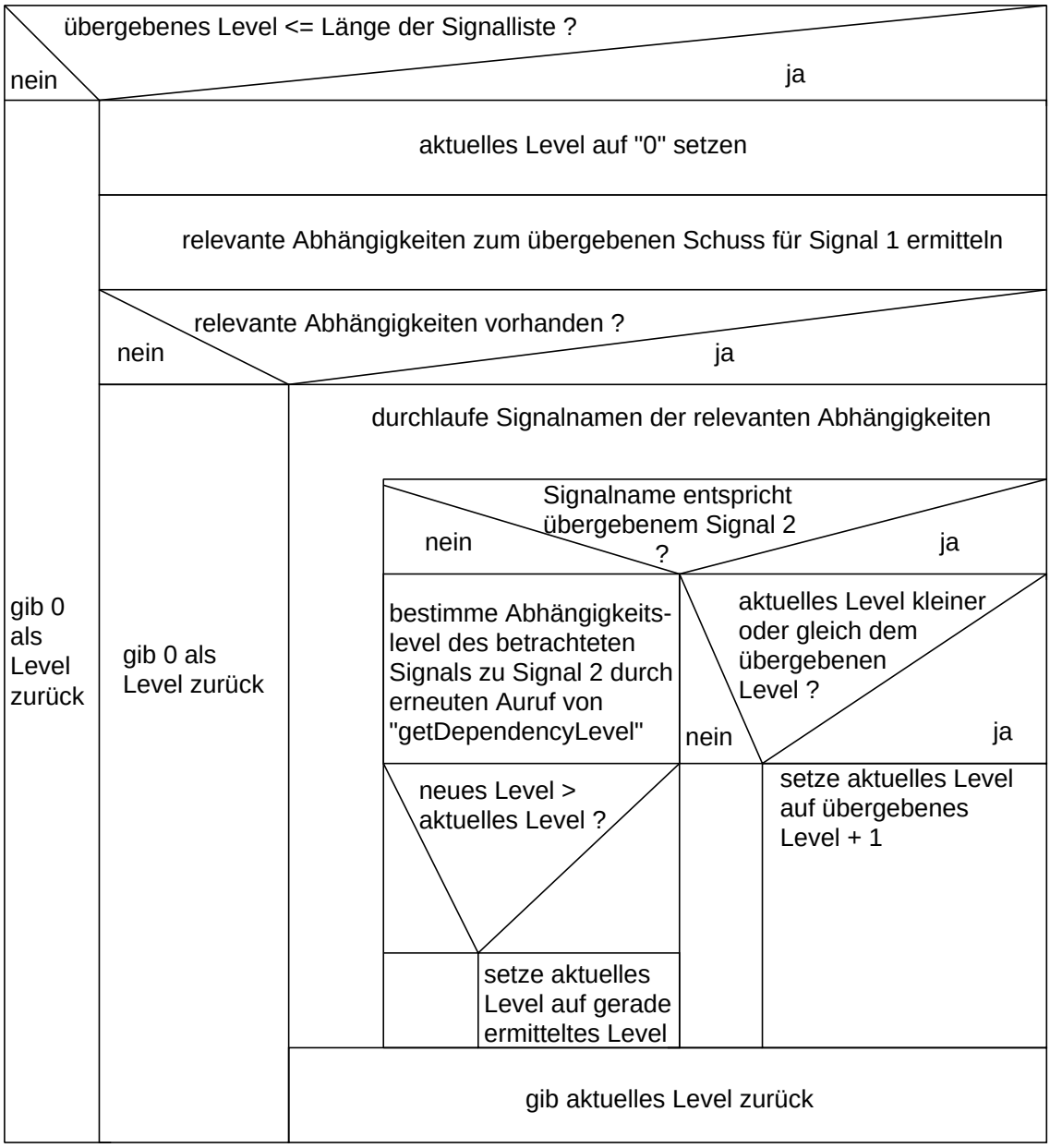


Abbildung 4.12: Rekursive Bestimmung eines Abhängigkeitslevels

Stellt man die betrachteten Signalabhängigkeiten wieder in Form eines Baumes dar, so wird deutlich, dass das beschriebene Vorgehen dem in der Informatik als ”Tiefensuche” bezeichneten Verfahren zum Durchlaufen eines Graphen entspricht.
Bei der Tiefensuche (engl. ”DFS - depth-first-search”) geht man so weit wie möglich einen gewählten Pfad entlang. Ist man am Ende eines Zweiges angekommen, geht man schrittweise zurück, bis man in einen bislang unbesuchten Teilbaum absteigen kann. Dies wird solange durchgeführt, bis keine unbesuchten Knoten mehr vorhanden sind.
Der in Abbildung 4.13 dargestellte Baum verdeutlicht das Vorgehen für das Bestimmen eines Abhängigkeitslevels für das Signal ”PTOT” zum Signal ”NELFJC” im Schuss 96518. Die Nummerierung der Kanten gibt die Reihenfolge beim Durchlaufen der Pfade in der durchgeführten Tiefensuche an.

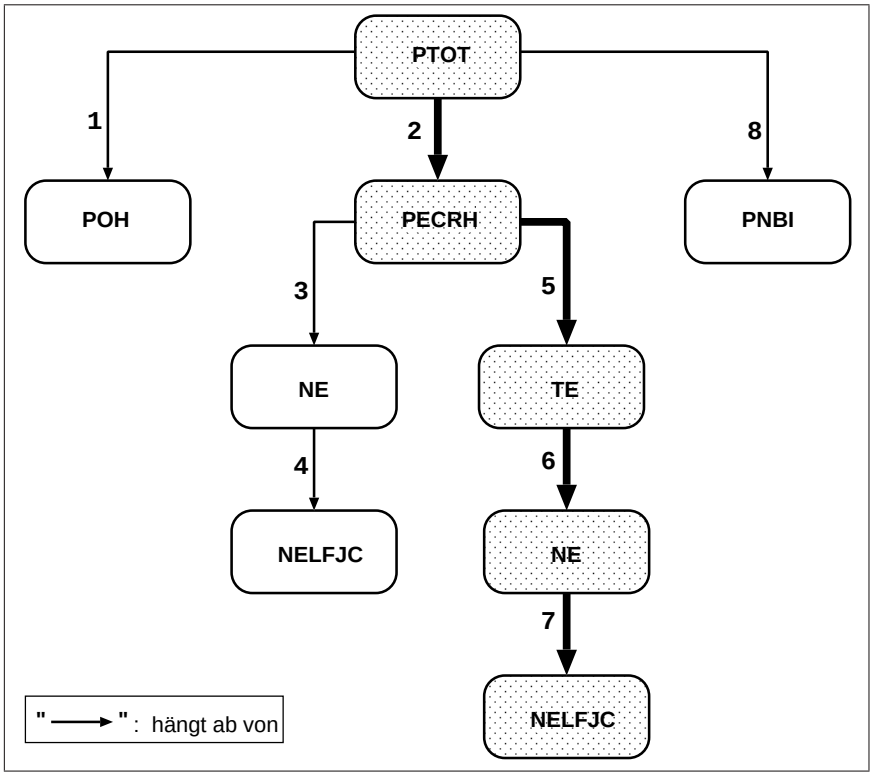


Abbildung 4.13: Tiefensuche zur Bestimmung des Abhängigkeitslevels

Der für das Abhängigkeitslevel ausschlaggebende Pfad von der Wurzel des Baumes (dem Signal ”PTOT”) zum Signal ”NELFJC” ist hervorgehoben. Wie zuvor beschrieben, wird das Abhängigkeitslevel aus dem längsten gefundenen Weg bestimmt. In diesem Fall beträgt das Abhängigkeitslevel für das Signal ”PTOT” also 4.

Die in der rekursiven Funktion ”getDependencyLevel” erzeugten Programmausgaben in Tabelle 4.14 entsprechen der Reihenfolge beim Durchlaufen der Pfade im zuvor dargestellten Baum. Das angezeigte Level entspricht dem Eingangsparameter ”v_level”.

```
...
*** Entering 'getDependencyLevel' with:
Signal 1: ptot
Signal 2: nelfjc
Level: 0

*** Entering 'getDependencyLevel' with:
Signal 1: poh
Signal 2: nelfjc
Level: 1

*** Entering 'getDependencyLevel' with:
Signal 1: pecrh
Signal 2: nelfjc
Level: 1

*** Entering 'getDependencyLevel' with:
Signal 1: ne
Signal 2: nelfjc
Level: 2

*** Entering 'getDependencyLevel' with:
Signal 1: te
Signal 2: nelfjc
Level: 2

*** Entering 'getDependencyLevel' with:
Signal 1: ne
Signal 2: nelfjc
Level: 3

*** Entering 'getDependencyLevel' with:
Signal 1: pnbi
Signal 2: nelfjc
Level: 1
...
```

Abbildung 4.14: Ablauf der rekursiven Bestimmung des Abhängigkeitslevels

Nach dem Bestimmen der Abhängigkeitslevel für die einzelnen Signale steht für jedes Signal aus der durch das Abfragen der Tabelle ”TPD_STORE” ermittelten Signalliste die Information zur Verfügung, ob und wann das Signal neu berechnet werden muss. Für den Schuss 96518 und die Suche nach Abhängigkeiten zum Signal ”NELFJC” ergeben sich die in Tabelle 4.9 dargestellten Abhängigkeitslevel.

Signal	**Abhängigkeitslevel**
NELA	1
NE	1
POH	0
HCNI10	0
HCNI20	0
...	...
HCNI90	0
TE	2
PNBI1	0
PNBI2	0
PRAD	0
EDIA	0
BETAT	0
BETAN	0
BETAP	0
PTOT	4
IECCD	3
PECRH	3
ECCDMAXR	3
ECRHPR	3
ECCDJR	3
PNBI	0
NELFJC	0

Tabelle 4.9: Abhängigkeiten zum Signal ”NELFJC” im Schuss 96518

Alle Signale mit einem Abhängigkeitslevel größer Null müssen neu berechnet werden. Da die Liste (in der Regel) unsortiert ist, wird zunächst das höchste Abhängigkeitslevel bestimmt und anschließend werden in einer Schleife von 1 bis zu diesem Maximum die entsprechenden Berechnungsprogramme angestoßen. Auf diese Weise wird die richtige Reihenfolge der Neuberechnungen eingehalten.

4.2.4 Berechnung einer neuen Signalversion

Zur Berechnung einer neuen Signalversion zu einer bestimmten Schussnummer werden die vorhandenen Intershot-Skripte und Programme verwendet, die auch im Rahmen der Chain1 - Ausführung genutzt werden. Diese müssen aus der PL/SQL-Prozedur ”solveDependencies” heraus angestoßen werden. Der Name des Programms, mit dem ein Signal erzeugt wurde, ist in der Beschreibungstabelle vermerkt. Es ist jedoch zu berücksichtigen, dass es Programme gibt, die mehrere Signale erzeugen. Damit keine doppelten Neuberechnungen eines Signals durchgeführt werden, wird jede Ausführung eines Programms vermerkt und es werden nur diejenigen Programme angestoßen, die zuvor noch nicht ausgeführt wurden.

Das Oracle®-Datenbanksystem stellt das Paket ”DBMS.SCHEDULER” bereit, welches die notwendige Funktionen und Prozeduren für das Erzeugen und Ausführen von Datenbank-Jobs enthält. Bei einem Datenbank-Job handelt es sich um die Kombination aus einem Programm und einem Ausführungsplan. Das Programm beschreibt die auszuführenden Aktionen und der Ausführungsplan legt fest, wann und in welchem Wiederholungsinterval das Programm gestartet wird. Mit diesen Datenbank-Jobs lassen sich neben PL/SQL-Code und -Funktionen auch externe Programme und Shell-Skripte aufrufen.
Der Zeitpunkt der Ausführung sowie eventuelle Wiederholungen lassen sich über die Attribute des Datenbank-Jobs steuern. Für die hier durchzuführende Signalerzeugung reicht eine einmalige Ausführung ohne zeitliche Verzögerung aus. Es besteht die Möglichkeit, den externen Programmen über den erzeugten Datenbank-Job Parameter für die Ausführung zu übergeben.

Für die Signalberechnung wird das vorhandene Shell-Skript ”intershot” verwendet. Dieses erzeugt zu einer vorgegebenen Schussnummer und mit einem vorgegebenen Programm eine neue Signalversion und fügt die Daten durch den Aufruf weiterer Skripte in die entsprechenden Tabellen der TPD ein. Als Parameter an das Skript müssen also beim Aufruf aus PL/SQL heraus die Schussnummer und der Name des Programms, mit dem das entsprechende Signal erzeugt werden kann, übergeben werden. Die in Abbildung 4.15 dargestellte Prozedur ”callIntershot” erzeugt einen entsprechenden Datenbank-Job namens ”intershot_calculation”. Dieser ruft das Shell-Skript ”intershot” mit einer Schussnummer und einem Programmnamen als Parameter auf. Der erzeugte Job wird innerhalb der Prozedur ausgeführt und anschließend gelöscht. Da es jeweils nur einen Datenbank-Job mit demselben Namen geben darf, wird zu Beginn der Prozedur geprüft, ob ein entsprechender Job bereits existiert und dieser ggf. entfernt. Dies ist ein weitere Grund dafür, dass mehrfache (gleichzeitige) Ausführungen der Prozedur ”solveDependencies” unbedingt verhindert werden müssen. Bei einem zweiten Aufruf der Prozedur ”callIntershot” würde ein durch einen vorherigen Aufruf gestarteter Datenbank-Job und somit die Neuberechnung eines Signals abgebrochen werden.

```
PROCEDURE callIntershot (
v_shot_number IN NUMBER,
v_prog_name IN VARCHAR2)
IS
...
BEGIN
   IF jobExists ('intershot_calculation') THEN
     ...
     dbms_scheduler.drop_job('intershot_calculation');
   END IF;

   dbms_scheduler.create_job(
     job_name => 'intershot_calculation',
     job_type => 'executable',
     job_action => '/home/chain1/Chain1/Intershot/intershot',
     number_of_arguments => 2,
     enabled => FALSE,
     auto_drop => TRUE
   );

   dbms_scheduler.set_job_argument_value (
     job_name => 'intershot_calculation',
     argument_position => 1,
     argument_value => v_shot_number
   );

   dbms_scheduler.set_job_argument_value (
     job_name => 'intershot_calculation',
     argument_position => 2,
     argument_value => v_prog_name
   );

   dbms_scheduler.run_job('intershot_calculation');
   dbms_scheduler.drop_job('intershot_calculation');

END callIntershot;
```

Abbildung 4.15: PL/SQL-Prozedur zur Ausführung eines Datenbank-Jobs

4.2.5 Erzeugen von Metainformationen

Zu den automatisch neu berechneten Signalversionen werden über die Intershot - Programme automatisch wichtige Metainformationen wie Versionsnummer, Erstellungsdatum, benutztes Programm etc. in der zugehörigen Beschreibungstabelle eingefügt. Es soll jedoch zudem vermerkt werden, warum die jeweilige Version eines Signals erzeugt wurde. Diese zusätzlichen Informationen zu den Signalen sollen in die TPD integriert werden.

Da es für alle Signale in der TPD bereits zu jedem Schuss und jeder Signalversion einen Eintrag in der entsprechenden Beschreibungstabelle gibt, erfolgt das Hinzufügen der zusätzlichen Versionsinformation durch Ergänzen der Beschreibungstabellen um eine weitere Tabellenspalte. Dies hat den Vorteil, dass keine weiteren Tabellen erzeugt und verwaltet werden müssen. Zudem wird so die Zuordnung der Informationen zu der jeweiligen Schuss- und Versionsnummer sichergestellt, ohne Daten mehrfach speichern zu müssen. Allerdings müssen auf Grund der Änderung der Tabellenstrukturen die Intershot-Programme und Skripte angepasst werden, die das Einfügen von Daten in die TPD durchführen (z.B. die Controlfiles für den SQL-Loader).

In der zusätzlichen Tabellenspalte kann bei automatisch berechneten Signalversionen, die auf Grund von Abhängigkeiten in der TPD erzeugt wurden, mit Hilfe der PL/SQL-Prozdur ”solveDependencies” die Abhängigkeit zu dem zuvor neu berechneten Signal als Grund für die Erstellung der neuen Signalversion automatisch eingetragen werden.

Hierzu wurde eine weitere PL/SQL-Prozedur ”insertMetaData” implementiert, die von der Prozedur ”solveDependencies” nach der Durchführung einer automatischen Signalberechnungen aufgerufen wird. Diese generiert an Hand der Signalnamen des abhängigen Signals und des Signals, zu dem die Abhängigkeit besteht, sowie der Schussnummer ein entsprechendes SQL-Statement und führt dieses aus. Der Auszug aus der Tabelle ”NE_DESCRIPTION” in Tabelle 4.10 zeigt ein Beispiel für den Eintrag der zusätzlichen Metainformationen für das Signal NE. In diesem Fall wurde nach der manuellen Neuberechnung des Signals ”NELFJC” automatisch einen neue Version des Signals ”NE” erstellt und eine entsprechende Information vermerkt.

shtid	**version**	**vers_reason**
...	...	...
95687	2	dependency on signal nelfjc
...	...	...

Tabelle 4.10: Beispieleintrag neuer Metainformationen in der Beschreibungstabelle des Signals ”NE”

4.3 Kapitelzusammenfassung

Die automatische Aktualisierung abhängiger Signale wird durch die Verwendung von Datenbank-Triggern und den Aufruf einer PL/SQL-Prozedur implementiert. Dabei muss die Vermeidung von Kollisionen in Form von mehrfachen Prozeduraufrufen und parallelen Tabellenzugriffe verhindert werden. Die Auflösung der Signalabhängigkeiten erfolgt durch rekursive Auswertung der Beschreibungstabellen zu den in Frage kommenden Signalen. Die notwendigen Neuberechnungen der abhängigen Signale werden über einen Datenbank-Job und den Aufruf der zur Verfügung stehenden Berechnungsprogramme durchgeführt. Anschließend werden zusätzliche Metainformationen zu den neu erstellten Signalversionen in die TPD eingefügt.

Kapitel 5

Darstellung zusätzlicher Metainformationen

Wie in Kapitel 4.2.5 auf Seite 50 beschrieben, werden bei der automatischen Berechnung neuer Signalversionen zusätzliche Metainformationen in die TPD eingefügt. Diese sollen den wissenschaftlichen Mitarbeiten zugänglich gemacht werden. Daher werden im Folgenden die Anforderungen an die Darstellung dieser Metainformationen sowie die konkrete Implementierung der Anzeige beschrieben.

5.1 Anforderungen

Die Metainformationen zu den einzelnen Signalversionen sollen als Versionshistorie verfügbar gemacht werden. Dabei sollen für die einzelnen Signale für jede Signalversion zu einem bestimmten Schuss jeweils die Versionsnummer, das Datum der Erstellung und der Grund für die Berechnung angezeigt werden. Die Anzeige einer Versionshistorie ist dabei in das TWU zu integrieren, wo sich neben den Signaldaten bereits, wie in Abbildung 5.1 auf der nächsten Seite zu sehen, die übrigen Metainformationen, die in den Beschreibungstabellen abgelegt sind, abrufen lassen.
Für die Anzeige von Signaldaten aus der TPD und zugehöriger Signaleigenschaften auf Basis der Bulk- und Description-Tabellen exisitiert bereits ein PL/SQL-Package ”tpd”. Um die Einbindung einer Versionshistorie möglichst einfach zu halten, soll dieses Package daher entsprechend erweitert werden.

5.2 Implementierung

Damit die Wissenschaftler die entsprechenden Informationen per Browser abrufen können, wird für die Darstellung der Metainformationen als Webseite im TWU das PL/SQL

Abbildung 5.1: Abrufen von Signalinformationen über das TWU

Web Toolkit verwendet, welches in der für die TPD verwendeten Oracle® -Datenbank enthalten ist. Dieses Toolkit ermöglicht das Generieren von Webseiten (HTML-code) aus PL/SQL-Prozeduren heraus. Um die erzeugten Webseiten den Clients bereitzustellen, wird zusätzlich ein Web Application Server benötigt, der die erzeugten Webseiten zum Browser des Benutzers schickt. Im TEXTOR-Umfeld wird hierzu ein Oracle® Application Server 10g eingesetzt, welcher neben einem Apache Webserver[1] das spezielle Modul ”mod_plsql”, ein sogenanntes ”PL/SQL-Gateway” enthält. Über diese PL/SQL-Gateway kann der Apache Webserver direkt mit der PL/SQL-Engine in der Oracle® Datenbank kommunizieren. Abbildung 5.2 auf der nächsten Seite zeigt den Abblauf bei der Bereitstellung von Webseiten über das PL/SQL Web Toolkit.

[1] Apache Webserver: frei verfügbarer und weit verbreiteter Webserver der Apache Software Foundation

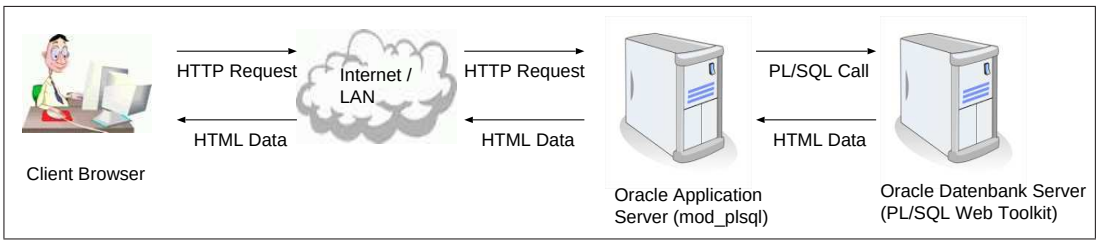


Abbildung 5.2: Abrufen von Signalinformationen über das TWU

Für die Verknüpfung von virtuellen Pfaden auf dem Webserver, die über spezielle URLs angesprochen werden können, mit der zu verwendenden Datenbankverbindung wird innerhalb des Application Servers ein sogenannter ”Database Access Descriptor” erstellt. Dort wird das Datenbankschema eingetragen, in dem die PL/SQL-Prozeduren und - Funktionen für die Erzeugung der Webseiten abgelegt sind. Der Name der zu verwendenden Prozedur oder Funktion wird in der aufgerufenen URL angegeben. Eine entsprechende URL kann z.B. wie folgt aussehen:

```
http://hostname/pls/DADname/<package>.<procedure>
```

Der `hostname` spezifiert dabei den Host, auf dem der Oracle® Application Server läuft. Die Pfadangabe `pls` sorgt dafür, dass der Apache Webserver die Anfrage an das Modul ”mod_plsql” weiterleitet. Der `DADname` stellt die zu verwendende Datenbankverbindung bereit, über welche die Prozedur `<procedure>` aus dem Paket `<package>` aufgerufen wird.
Zur Gestaltung der Webseite stellt das PL/SQL Web Toolkit vorgefertigte Prozeduren und Funktionen wie ”htp.htmlopen” zum Öffnen einer neuen Webseite oder ”htp.print” für die Ausgabe von Text innerhalb der Webseite bereit. Beim Aufruf einer solchen Webseite ist es möglich, durch Angaben in der aufgerufenen URL Parameter an die dahinterliegende Prozedur oder Funktion zu übergeben. Bei der Darstellung der Versionsinformationen werden auf diese Weise die Schussnummer und der Name des Signals, für das die Informationen angezeigt werden sollen, an die zuständige Prozedur übergeben. Die Verwendung des PL/SQL Web Toolkit wird in [LIT14, Kapitel 7] ausführlich beschrieben.

Die in das Package ”tpd” eingefügte Prozedur ”version_history” besitzt die beiden Parameter ”v_shtid” und ”v_signal_name”. An Hand dieser Angaben wird die Beschreibungstabelle des entsprechenden Signals abgefragt und es werden die benötigten Informationen (Signalversion, Datum der Erstellung, Grund für die Erstellung) zu dem vorgegebenen Schuss abgefragt. Aus den Informationen wird dann mit Hilfe der oben beschriebenen

Prozeduren und Funktionen aus dem PL/SQL Web Toolkit und unter Verwendung der beiden Hilfsprozeduren ”twuHTMLhead” und ”twuHTMLtail” eine Webseite generiert. Abbildung 5.3 skizziert das Vorgehen bei der Darstellung der Versionsinformationen innerhalb der Prozedur ”version_history”.

```
procedure version_history (v_shtid in NUMBER,
v_signal_name in VARCHAR2 )
is

...

begin
   twuHTMLhead (2, 'Version History of '|| v_signal_name
   || ' for ' || v_shtid, v_signal_name);
   htp.print ('<br>');
   htp.print ('<pre>');
   htp.print ('Version Date Reason for Calculation<br>');

   v_query := 'SELECT version, crea_date, vers_reason FROM ' ||
              v_table_name || ' WHERE shtid = ' || v_shtid ||
              ' ORDER BY version ASC';

   OPEN vers_cursor FOR v_query;
   LOOP
     FETCH vers_cursor INTO v_vers_row;
     EXIT WHEN vers_cursor%NOTFOUND;
     htp.print(v_vers_row.version || ' '
               || to_char(v_vers_row.crea_date, 'yyyy/mm/dd')
               || ' ' || v_vers_row.vers_reason);
   END LOOP;
   htp.print ('</pre>');
   twuHTMLtail (2);
end version_history;
```

Abbildung 5.3: Prozedur ”version_history” zur webbasierten Darstellung von Versionsinformationen

Um die Darstellungsform mit der Anzeige der übrigen Signaleigenschaften und -daten gleichzuhalten, wurde auf HTML-Formatierungen wie die Verwendung einer Tabelle verzichtet, und stattdessen die <pre>-Umgebung benutzt, mit der ein vorformatierter Text genau so im Browser dargestellt wird, wie er im Quelltext angegeben wurde.

Abbildung 5.4 zeigt die Versionsinformationen für das Signal ”ZEFFAVG” im Schuss 95010 sowie die zugehörige URL, über die sich die Informationen abrufen lassen. In der URL sind die Parameter ”v_signal_name” und ”v_shtid” enthalten, die über das PL/SQL-Modul im Webserver an die aufgerufene Prozedur ”version_history” übergeben werden.

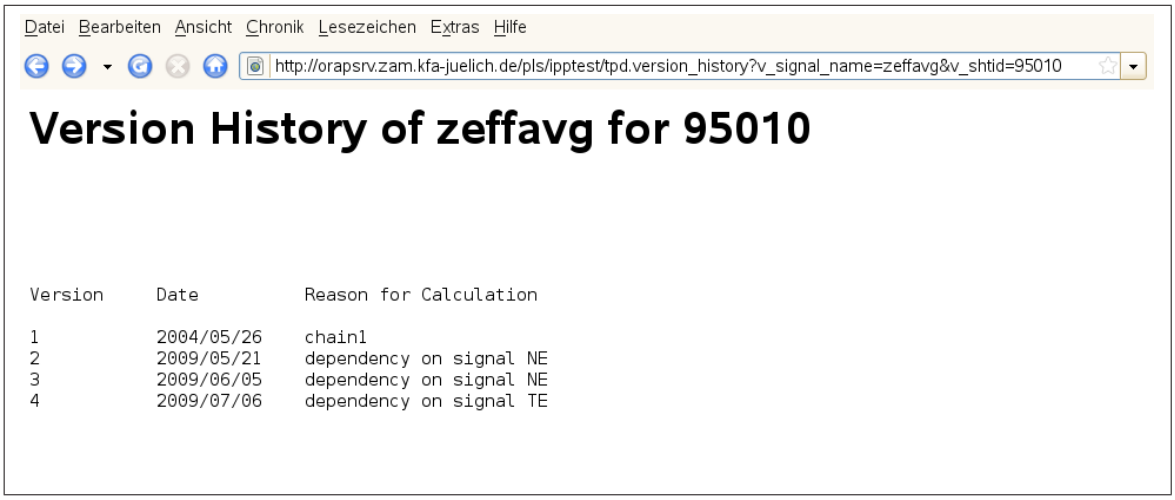


Abbildung 5.4: Beispiel für die Ausgabe einer Versionshistorie

5.3 Kapitelzusammenfassung

Die zusätzlich eingfügten Metainformationen zu den automatisch neu erstellen Signalversionen werden in Form einer Versionshistorie als Webseite im TWU verfügbar gemacht. Hierzu wird das PL/SQL Web Toolkit verwendet. Das Package zur Darstellung der übrigen Meta- und Signaldaten wurde um eine entsprechende Prozedur ”version_history” erweitert.

Kapitel 6

Verbesserung der Performance von Signalabfragen

Dieses Kapitel beschäftigt sich mit der möglichen Opitimierung der auftretenden Datenbankabfragen und stellt die verfolgten Ansätze sowie die Ergebnisse durchgeführter Zeitmessungen dar. Als erstes werden dabei die zu untersuchenden Abfragen vorgestellt und die Testumgebung sowie das Vorgehen bei den Zeitmessungen beschrieben. Anschliessend werden die verschiedenen Optimierungsansätze sowie die Ergebnisse der jeweiligen Zeitmessungen dargestellt.

6.1 Problemstellung und Anforderungen

Die Abfragezeiten für Signale aus der TPD spielen eine wichtige Rolle bei der Auswertung des TEXTOR-Experiments. Ein effizientes Arbeiten mit den Signaldaten setzt voraus, dass das Abfragen der Signale aus der Datenbank ohne lange Wartezeiten erfolgen kann. Die Größe der Signaltabellen und das Auftreten unterschiedlicher, zum Teil sehr komplexer Abfragen erschwert es jedoch, die Abfragezeiten gering zu halten. Das in Kapitel 3.3.2 auf Seite 22 beschriebene TWU bietet einen Web-basierten Zugriff auf Rohdaten im Common Storage Facility und Signaldaten aus der TPD. Hierbei können über Links innerhalb der Weboberfläche bestimmte Schussnummern und Signale ausgewählt und die zugehörigen Daten angezeigt werden. Neben dem Zugriff auf Signaldaten über das TWU wird den Wissenschaftlern eine weitere Weboberfläche bereitgestellt, der ”TPD Shot Selector”. Hierbei handelt es sich um ein Java-basiertes Werkzeug, mit dem sich Schüsse mit bestimmten Eigenschaften suchen lassen. Zur Abfrage sind keine SQL-Kenntnisse notwendig, da die Anfragen an die Datenbank automatisch aus den Formulareingaben generiert und ausgeführt werden. Abbildung 6.1 zeigt die Weboberfläche des TPD Shot Selector.

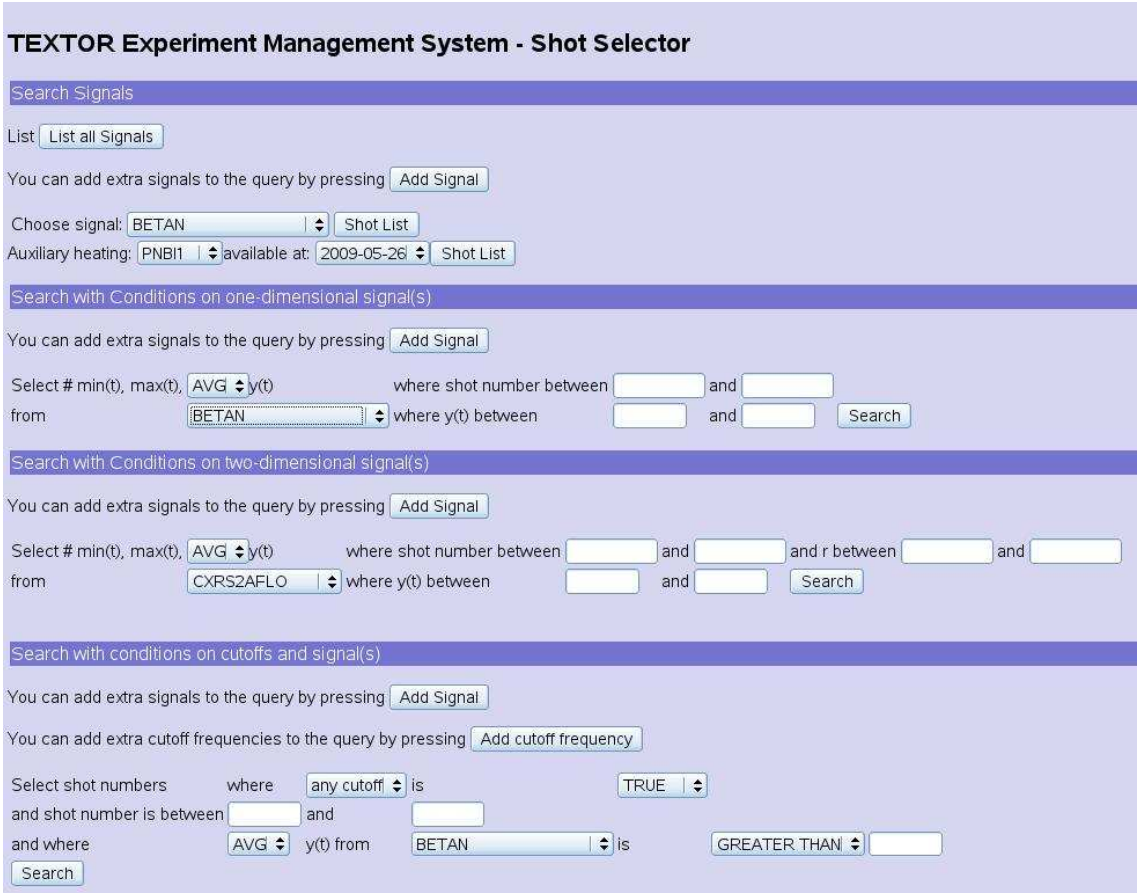


Abbildung 6.1: TPD Shot Selector

Im Abschnitt ”Search Signals” lässt sich eine Liste aller TPD-Signale anzeigen sowie nach Schüssen suchen, bei denen bestimmte Signale berechnet wurden bzw. ein bestimmtes Heizsystem eingesetzt wurde. In den beiden folgenden Abschnitten der Oberfläche kann nach Schüssen gesucht werden, bei denen die Minimum-, Maximum oder Durchschnittswerte vorgegebener Signale in einem bestimmten Bereich liegen. Im letzten Abschnitt besteht die Möglichkeit, nach dem Auftreten von sogenannten ”Cutoffs” zu suchen. Dabei handelt es sich um ein physikalisches Phänomen in der Kernfusion, das bei der Ausbreitung von Elektronen-Zyklotron-Strahlung im Plasma auftreten kann. Durch einen Cutoff wird die Mikrowellen-Strahlung so stark gedämpft, dass die Temperaturmessung nicht mehr möglich ist.

Die an Hand der Formulareingaben erzeugten SQL-Abfragen können sehr umfangreich werden, besonders wenn mehrere Signale und Bedingungen berücksichtigt werden sollen. Im Gegensatz zu den Abfragen im TWU, bei denen in der Regel nur die Daten eines

Signals zu einem bestimmten Schuss abgerufen werden, kann die Ausführung der Abfragen im TPD Shot Selector daher sehr lange (mehrere Minuten) dauern und verhindert dadurch eine effiziente Auswertung.
Die Problematik langer Datenbankabfragen wird zudem dadurch verschärft, dass der Webserver und die Datenbank einen Mehrbenutzerbetrieb ermöglichen und deshalb unter Umständen mehrere Anfragen parallel verarbeitet werden müssen.

6.2 Testumgebung und Zeitmessung

Die Untersuchungen zur Verbesserung der Abfragezeiten von TPD-Signalen werden auf einem Testsystem durchgeführt. Dadurch soll einerseits eine zusätzliche Belastung des Produtkionssystems verhindert und gleichzeitig eine Verfälschung der Zeitmessungen durch parallele Abfragen aus dem Produktionsbetrieb verhindert werden.
Da eine Migration des TPD-Datenbankservers auf eine neue Hardware sowie ein neues Betriebssystem geplant ist und die neue Hardware bereits zur Verfügung steht, kann diese für den Aufbau einer Testumgebung verwendet werden. Tablle 6.1 zeigt die Kenndaten des Testsystems.

CPU-Anzahl	2
CPU-Typ	Intel XEON® Quadcore (2.33 GHz)
Arbeitsspeicher	16 GB
Plattenplatz (intern)	140 GB
Plattenplatz (extern)	2.1 TB auf Storage (RAID 5)
Hersteller	DELL
OS	Oracle Enterprise Linux® 5.3 64 Bit

Tabelle 6.1: Kenndaten des Testsystems

Da der Datenbankserver in der späteren Produktion mit dem Betriebssystem Oracle Enterprise Linux® in Version 5.3 (64 bit) und einer Oracle® - Datenbank in Version 10.2.0.3 betrieben werden soll, wurde diese Software auch für die Testumgebung verwendet. Um ein möglichst genaues Abbild der Produktionsumgebung und dadurch aussagekräftige Messungen zu erhalten, wurden die Struktur der Datenbank sowie die physikalische Speicherstruktur identisch zum Produktionsserver eingerichtet. Die eigentlichen Daten der TPD wurden durch Ausführen eines Exports und eines anschließenden Imports vom Produktionssystem auf die Testumgebung übertragen.

Bei den Zeitmessungen sind einige Randbedingungen zu berücksichtigen. Weil bei den Performance-Untersuchungen die Datenbank im Mittelpunkt stehen soll, müssen Einflussfaktoren wie die Datenübertragung über das Netzwerk und der Aufbau der Ergebnisseiten durch die Java-Anwendung im TPD Shot Selector soweit wie möglich ausgeschlossen werden. Daher werden die zu untersuchenden Datenbankabfragen direkt auf dem Datenbankserver ausgeführt und dort die jeweiligen Zeiten gemessen.
Nur durch mehrfache Wiederholungen von Messungen zu den einzelnen Abfragen und durch das Ermitteln von Durchschnittswerten lassen sich Auswirkungen interner Prozesse der Datenbankverwaltung und des automatischen Zwischenspeicherns von Ergebnissen (Caching) innerhalb der Datenbank auf die Zeitmesssungen relativieren und so aussagekräftige Abfragezeiten bestimmen. Deshalb sollte sich die Zeitmessung der Abfragen leicht automatisieren und wiederholen lassen. Ein manuelles wiederholtes Ausführen der vielen verschiedenen Abfragen wäre zu aufwendig.

Es gibt verschiedene Möglichkeiten, die Zeit für die Ausführung einer Datenbankabfrage zu messen. Die von Oracle® bereitgestellten Programme SQL*Plus und SQLDeveloper, mit der sich SQL- und PL/SQL-Befehle ausführen lassen, zeigen nach der Ausführung eines Kommandos, wie in den Abbildungen 6.2 und 6.3 zu sehen, automatisch die benötigte Zeit an.

```
SQL> SELECT count(*)
2 FROM hcni30_bulk
3 WHERE shtid BETWEEN 94100 AND 95500
4 AND version = (SELECT max(version) FROM hcni50_bulk
5 WHERE shtid BETWEEN 94100 AND 95500);

COUNT(*)
-----
74286313

Elapsed: 00:00:50.34
```

Abbildung 6.2: Zeitmessung in SQL*Plus

Auf diese Weise lassen sich jedoch zunächst nur die Zeiten einzelner Abfragen messen. Für eine Automatisierung und die wiederholte Ausführung zum Erstellen von Mittelwerten müssten die Abfragen z.B. in PL/SQL - Code eingebettet werden. Da PL/SQL jedoch

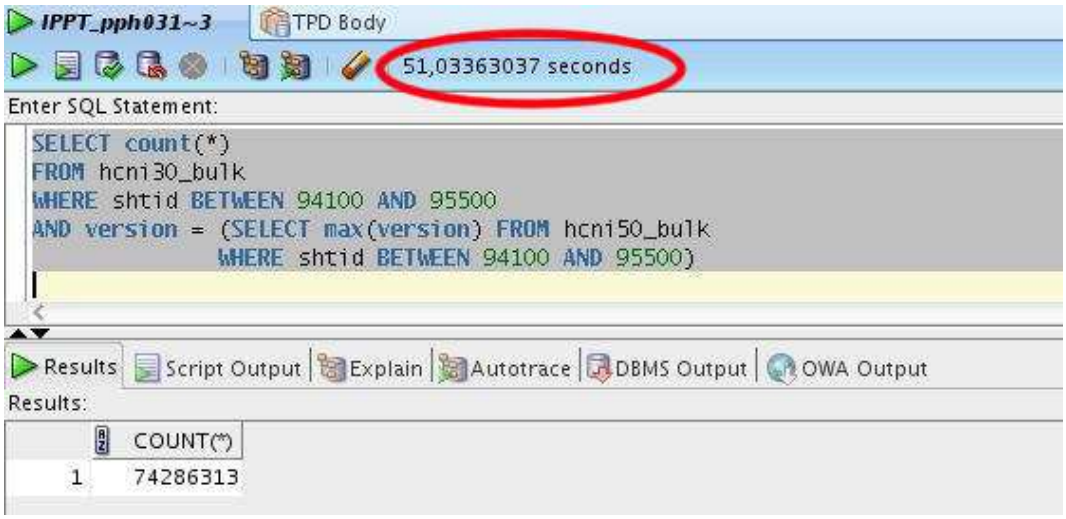


Abbildung 6.3: Zeitmessung im SQLDeveloper

keine hauseigenen Funktionen für die Zeitmessung von Abfragen bereitstellt, müssten die einzelnen Zeiten umständlich aus der Zeit für die Ausführung des gesamten Skripts, die der SQLDeveloper oder SQL*Plus zurückgeben, ermittelt werden.

Eine weitere Möglichkeit für das Ermitteln von Abfragezeiten ist die Verwendung der von Oracle® bereitgestellten Tools ”explain plan” und ”autotrace”. Mit diesen Tools lässt sich das Vorgehen bei der Ausführung eines Statements detailliert darstellen. Die Abbildung6.4 zeigt ein Beispiel für die Nutzung von ”explain plan”. Die Ausgabe von ”autotrace” (erhältlich über die Opion ’set autotrace traceonly’ in SQL*Plus) ist nahezu identisch. Sieh hierzu auch [LIT15, S. 974 - 983].
Nachteil bei dieser Vogehensweise ist jedoch, dass auf Grund der zusätzlich erzeugten Informationen die Abfragezeiten des Statements beeinflusst und unter Umständen verfälscht werden. Zudem ist eine Filterung der entsprechenden Abfragezeit für das ausgeführte Statement auf Grund der Fülle von Informationen ggf. sehr aufwendig.

Um die Anforderungen an die Zeitmessungen von TPD-Abfragen zu erfüllen, wird daher ein PHP-Skript verwendet, welches auf dem Datenbankserver ausgeführt wird. PHP ist eine frei verfügbare (Open Source) Skriptsprache, deren Syntax an die der Programmiersprache ”C” angelehnt ist. PHP wird hauptsächlich zur Erstellung von dynamischen Webseiten oder Webanwendungen verwendet. Über die in PHP zur Verfügung stehende Funktion ”microtime” lässt sich eine Zeitnahme sehr leicht durch das Ermitteln einer Start- und einer Endzeit vor bzw. nach dem Ausführen einer Abfrage implementieren. Zudem lassen sich die Messungen durch Verwendung von Funktionen und Schleifen gut

```
SQL> explain plan for
 2  SELECT count(*)
 3  FROM hcni30_bulk
 4  WHERE shtid BETWEEN 94100 AND 95500
 5  AND version = (SELECT max(version) FROM hcni50_bulk
 6  WHERE shtid BETWEEN 94100 AND 95500);

 Explained.

 SQL> select * from table(dbms_xplan.display());

 PLAN_TABLE_OUTPUT
 ------------------------------------------------------------------------
 Plan hash value: 3175805418
 ------------------------------------------------------------------------
 | Id  | Operation                     | Name            | Time     |
 ------------------------------------------------------------------------
 |   0 | SELECT STATEMENT              |                 | 00:34:05 |
 |   1 |  SORT AGGREGATE               |                 |          |
 |*  2 |   TABLE ACCESS BY INDEX ROWID | HCNI30_BULK     | 00:19:48 |
 |*  3 |    INDEX RANGE SCAN           | HCNI30_BULK_IDX | 00:05:33 |
 |   4 |    SORT AGGREGATE             |                 |          |
 |   5 |     TABLE ACCESS BY INDEX ROWID| HCNI50_BULK     | 00:14:18 |
 |*  6 |      INDEX RANGE SCAN         | HCNI50_BULK_IDX | 00:04:00 |
 ------------------------------------------------------------------------
```

Abbildung 6.4: explain plan - Beispiel

automatisieren und wiederholen. Für die Ausführung von Datenbankabfragen stehen in PHP verschiedene Funktionen (basierend auf OCI8[1]) bereit. Die gemessenen Zeiten werden in eine Datei ausgegeben, die zur späteren Auswertung verwendet wird. Da die auszuführenden SQL-Befehle automatisch aus einer Datei eingelesen werden, kann somit die gesamte Zeitmessung durch einen einzigen Aufruf des Php-Skiptes durchgeführt werden. Abbildung 6.5 skizziert das Vorgehen bei der in PHP implementierten Zeitmessung.

[1]Oracle Call Interface: Programmierschnittstelle für den Zugriff auf Oracle® -Datenbanken aus Programmiersprachen wie C, C++ oder PHP heraus

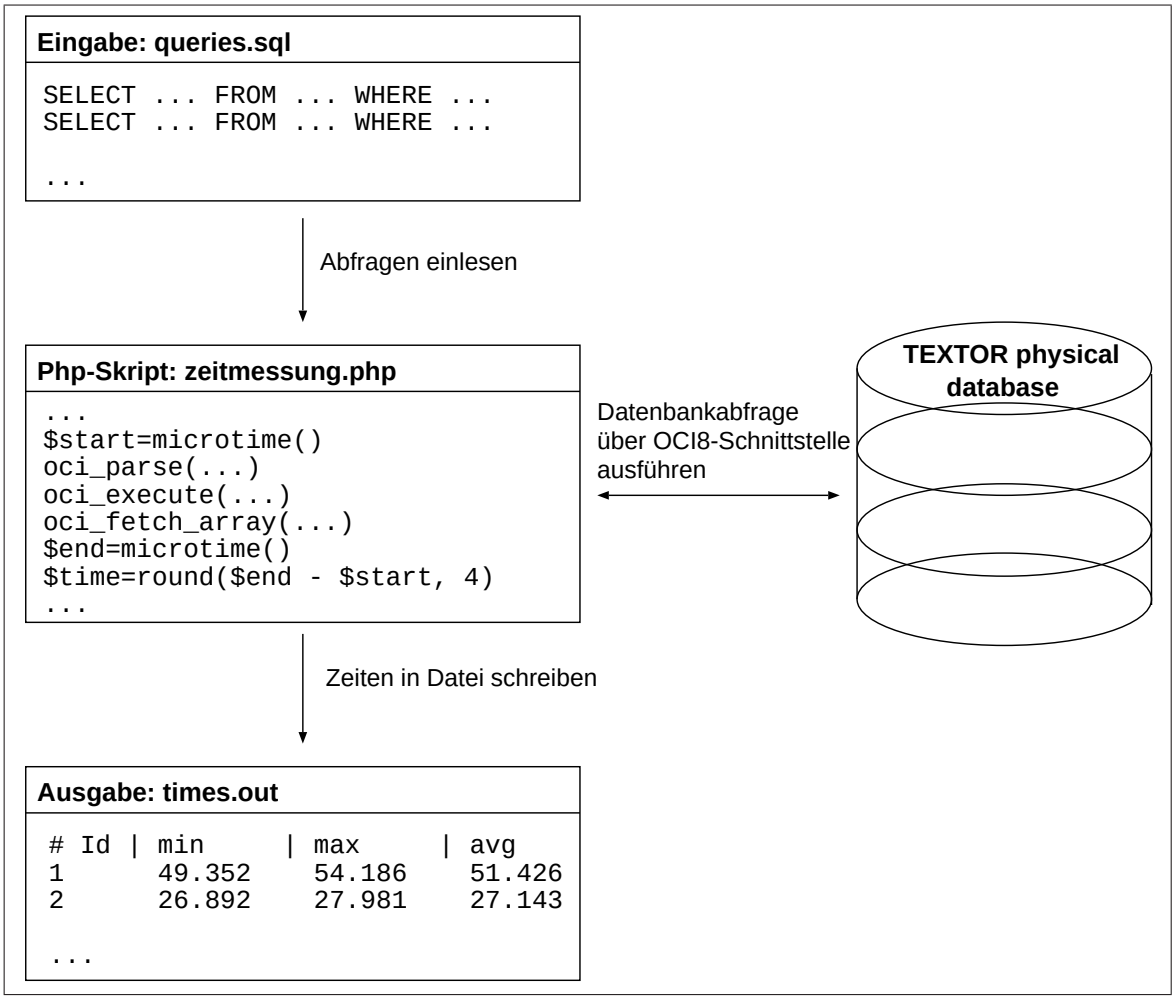


Abbildung 6.5: Durchführung der Zeitmessungen mit PHP

6.3 Partitionierung von Signaltabellen

Die Größe der vorhandenen Signaltabellen erschwert die Verwaltung der Daten und wirkt sich negativ auf die Abfragezeiten aus. Daher wird eine Aufteilung der Tabellen in Partitionen untersucht und geprüft, ob die Partitionierung zu einer Beschleunigung der Signalabfragen führt. Zunächst werden die Eigenschaften und das Vorgehen bei der Partitionierung von Tabellen dargestellt. Anschließend wird die Umsetzung für ausgewählte TPD-Tabellen innerhalb der Testumgebung beschrieben, bevor auf die durchgeführten Abfragen und die Ergebnisse der Zeitmessungen eingegangen wird.

6.3.1 Konzept der Partitionierung

Große Tabellen und Indizes in einer Oracle®-Datenbank lassen sich mit Hilfe des ”Partitioning” in kleinere, leichter zu verwaltende Einheiten aufteilen. Neben Vorteilen bei der Verwaltung der Objekte (z.B. beim Erzeugen von Sicherungen) kann das Partitioning auch die Geschwindigkeit von Abfragen der partitionierten Tabellen verbessern. Einzelne Partitionen können z.B. parallel durchsucht und aktualisiert werden. Zudem können Partitionen, die für eine Abfrage nicht berücksichtigt werden müssen, beim Ausführen der Abfrage übersprungen werden (dies wird ”partition pruning” genannt). Weiterhin können Join-Operationen (Verknüpfen der Informationen mehrerer Tabellen) durch die partitionsweise Ausführung von Joins ggf. optimiert werden.
Falls eine Partition nicht verfügbar ist, kann mit anderen Partitionen in der Regel weitergearbeitet werden. Das Partitioning ist für Anwendungen transparent, so dass die Tabellenabfragen nicht angepasst werden müssen. Abbildung 6.6 zeigt das Prinzip einer Tabellenpartitionierung.

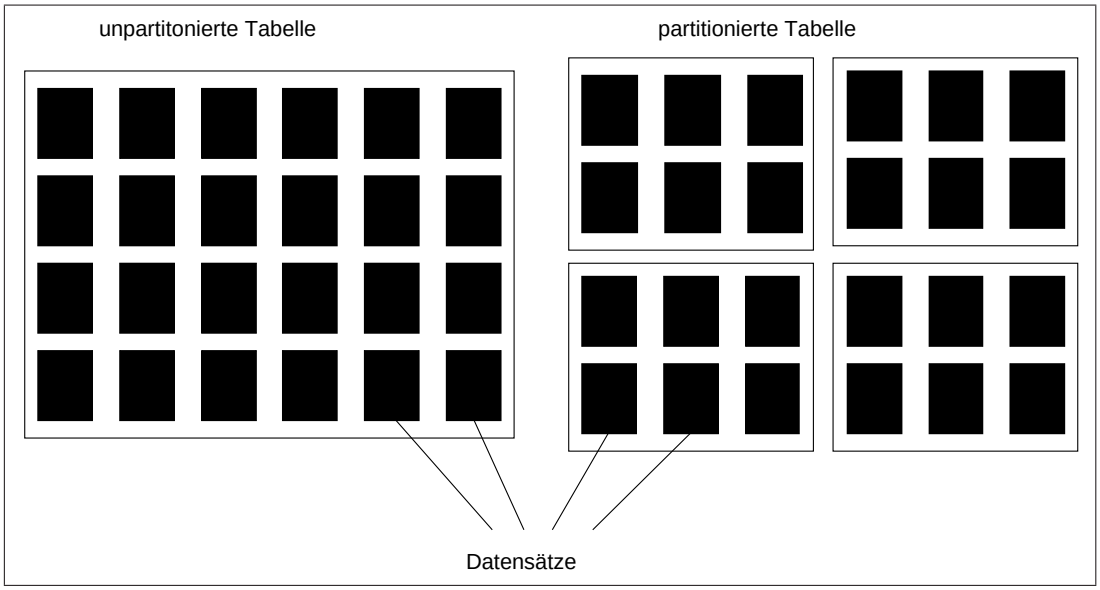


Abbildung 6.6: Partitionieren einer Tabelle

Für die Aufteilung der Objekte gibt es verschiedene Vorgehensweisen. Die am häufigsten verwendete Methode ist das ”Range-Partitioning”. Hierbei werden die Daten auf Grundlage von Bereichen von Partitionsschlüsselwerten den einzelnen Partitionen zugeordnet. Partitionsschlüssel können aus mehreren Tabellenspalten bestehen. Ein Beispiel für das Range-Partitioning wäre die Aufteilung einer Tabelle anhand einer Zeitspalte (Datumswerte) in verschiedene Zeitabschnitte (z.B. Jahre, Monate). Für jedes Jahr / jeden Monat

könnte dann eine eigene Partition erstellt werden. Der Partitionsschlüssel wäre in diesem Fall die Zeitspalte.

Um die Vorteile des Partitioning möglichst gut ausnutzen zu können, sollten die Partitionsgrößen annähernd gleich sein, also eine Gleichverteilung der Daten auf die einzelnen Partitionen angestrebt werden. Ist dies nicht der Fall, würden also im obigen Beispiel einem Zeitraum wesentlich mehr Tabellenzeilen zugeordnet als einem anderen, sollte über eine andere Aufteilung nachgedacht werden. So könnte dann z.B. das ”Hash-Partitioning” eingesetzt werden, welches einen Hash-Algorithmus[2] auf den Partitionsschlüssel anwendet und dadurch die Daten gleichmäßig auf die einzelnen Partitionen verteilt. Die Tabellenzeilen können allerdings auch an Hand einer Liste diskreter Werte für den Partitionsschlüssel zugeordnet werden. Diese Vorgehensweise wird ”List-Partitioning” genannt. Ein Beispiel hierfür wäre eine Tabelle mit Personendaten, die in einer der Spalten das Geschlecht (m oder w) vermerkt und in zwei Partitionen (Männer und Frauen) aufgeteilt wird. Kombinationen aus den einzelnen Vorgehensweisen sind ebenfalls möglich, etwa ein ”Range-Hash-Partitioning” oder ein ”Range-List-Partitioning”.

Das Oracle®-Datenbankmanagementsystem ermöglicht eine nachträgliche Bearbeitung von Partitionen. So kann z.B. eine vorhandene Partition in zwei Partitionen aufgeteilt werden (”split”). Abbildung 6.7 zeigt ein SQL-Beispiel für das Aufteilen einer Partition.

```
ALTER TABLE rechnungen
SPLIT PARTITION rech_q4
      AT (TO_DATE('16.11.2008','dd.mm.yyyy'))
INTO (PARTITION rech_q4_1
      TABLESPACE best_q4_1 STORAGE (INITIAL 4M NEXT 1M),
      PARTITION rech_q4_2
      TABLESPACE rech_q4_2 STORAGE (INITIAL 6M NEXT 1M));
```

Abbildung 6.7: Beispiel für die Aufteilung einer Tabellenpartition (”split”)

Umgekehrt lassen sich zwei Partitionen zu einer Partition zusammenfassen (”merge”). Abbildung 6.8 zeigt ein SQL-Beispiel für das Zusammenfassen zweier Partitionen.

Beim Partitioning sollte berücksichtigt werden, dass nach Möglichkeit auch die Indizes einer Tabelle entsprechend angepasst werden. Von Vorteil ist hierbei die Verwendung von

[2] Hash-Algorithmus: liefert eine (nahezu) eindeutige Kennzeichnung (”fingerprint”) zu einer größeren Datenmenge

```
ALTER TABLE rechnungen
MERGE PARTITIONS rech_q4_1, rech_q4_2
INTO PARTITION rech_q4,
TABLESPACE rech_q4
STORAGE (INITIAL 10M NEXT 2M);
```

Abbildung 6.8: Beispiel für das Zusammenfassen zweier Tabellenpartitionen (”merge”)

”local-prefixed” Indizes. Diese sind identisch zur Tabelle aufgeteilt und die führenden Indexspalten entsprechen denen des Partitionsschlüssels. Diese Form der Indizes bietet den Vorteil, dass ein Neuaufbau des Indexes (ein sogenannter ”rebuild”), der sehr zeitaufwendig und rechenintensiv sein kann, bei Änderungen an der Tabelle seltener notwendig ist als bei zusammenhängenden (”global”) Indizes.

6.3.2 Umsetzung innerhalb der TPD

Ziel ist es, die bei den durchgeführten Abfragen betrachteten Datenmengen zu reduzieren, indem nur die relevanten Teilbereiche einer Tabelle, also bestimmte Partitionen, berücksichtigt werden. Dies entspricht dem in Kapitel 6.3.1 beschriebenen ”partition pruning” und soll die Abfragezeiten verbessern.
Eine Partitionierung von Tabellen und Indizes bringt nur bei großen Objekten Vorteile. Auf der Testumgebung wurden daher für die Zeitmessungen einige der größeren Signaltabellen (mit Endung _BULK), die mehrere Millionen Zeilen enthalten, sowie deren Indizes in Partitionen aufgeteilt.

Die Schussnummern innerhalb der Signaltabellen stellen die Basis für die Verküpfung von Tabellen dar und sind sowohl bei der Signalabfrage über das TWU als auch bei der Nutzung des TPD Shot Selector Grundlage für die Selektion der Datenmenge. Deshalb wurde die Schussnummer (Spalte ”shtid”) als Partitionsschlüssel verwendet. Da beim TPD Shot Selector ausschließlich zusammenhängende Schussnummernbereiche angegeben werden können, müssen auch die einzelnen Partitionen der Signaltabellen zusammenhängende Schussnummernbereiche enthalten, um die Menge der bei einer Abfrage zu berücksichtigenden Tabellenpartitionen reduzieren zu können. Deshalb wird für die ausgewählten Signaltabellen das zuvor beschriebene Range-Partitioning verwendet, bei dem die einzelnen Partitionen jeweils einen zusammenhängenden Schussnummernbereich enthalten.
Die Verwendung eines Hash-Partitioning würde zwar eine gleichmäßige Verteilung von Tabellenzeilen auf die einzelnen Partitionen sicherstellen, könnte allerdings keine zusam-

menhängenden Schussnummernbereiche in den Partitionen realisieren.

Ein List-Partitioning ist nur bei einer kleinen Anzahl verschiedener Werte zu empfehlen und scheidet auf Grund der großen Anzahl unterschiedlicher Schussnummern ebenfalls aus.

Die Wahl gleich großer Schussbereiche für das Range-Partitioning führt leider nicht automatisch zu identischen Partitionsgrößen. Dies hängt damit zusammen, dass nicht zu jedem Schuss alle Signale berechnet werden und neue Signalversionen nicht zu allen Schüssen gleich häufig erstellt werden. Bei einem ersten Ansatz mit Schussbereichen der Größe 5000 enthielt die größte Partition 10 Mal so viele Zeilen wie die Kleinste.

Daher wurden für alle zu partitionierenden Signaltabellen die Schussbereiche so berechnet, dass alle Partitionen in etwa die gleiche Größe erhalten. Dabei wurde zunächst durch die in Abbildung 6.9 dargestellte Abfrage des Data Dictionaries die Anzahl der Zeilen in der unpartitionierten Tabelle ermittelt.

```
SELECT num_rows FROM user_tables
WHERE table_name = 'HCNI50_BULK'
```

Abbildung 6.9: Vorgehen beim Partitionieren der Tabelle ”hcni50_bulk” (1)

Anschließend wurde daraus zusammen mit der Anzahl zu erstellender Partitionen die Anzahl Zeilen ermittelt, die jeder Partition zugewiesen werden müssen. Bei der Anzahl der Partitionen pro Tabelle ist zu berücksichtigen, dass jede weitere Partition die Anzahl der Zeilen der einzelnen Partitionen reduziert, die Verwendung vieler kleiner Partitionen jedoch bei Abfragen großer Schussbereiche negative Auswirkungen auf die Performance haben könnte. Um eine deutliche Reduzierung der Zeilenanzahl von der Originaltabelle zu den einzelnen Partitionen zu erreichen und die Partitionsanzahl dennoch überschaubar zu halten, wurden hier die Tabellen jeweils in 10 Partitionen aufgeteilt. Bei einer Abfrage auf einen Schussbereich, der sich innerhalb einer Partition befindet, kann so die Menge der zu betrachtenden Daten im Vergleich zu der Originaltabelle ggf. auf ein Zehntel reduziert werden. Mit der in Abbildung 6.10 dargestellten Abfrage und der berechneten Zeilenanzahl pro Partition wurden die Obergrenzen der Schussbereiche der jeweiligen Partitionen ermittelt.

Abbildung 6.11 zeigt den PL/SQL-Code, mit dem die Tabelle ”hcni50_bulk” auf Basis der zuvor ermittelten Schussbereiche partitioniert werden kann.

```
SELECT max(shtid) FROM  (
  SELECT shtid FROM hcni50_bulk
  ORDER BY shtid
)
WHERE rownum < ... ;
```

Abbildung 6.10: Vorgehen beim Partitionieren der Tabelle ”hcni50_bulk” (2)

```
CREATE TABLE p_hcni50_bulk PARTITION BY RANGE(shtid)(
   PARTITION p_hcni50_bulk_1
     VALUES LESS THAN (90853)
     TABLESPACE p_signal_bulk_1,
   PARTITION p_hcni50_bulk_2
     VALUES LESS THAN (91459)
     TABLESPACE p_signal_bulk_2,
   PARTITION p_hcni50_bulk_3
     VALUES LESS THAN (95189)
     TABLESPACE p_signal_bulk_3,
   PARTITION p_hcni50_bulk_4
     VALUES LESS THAN (95892)
     TABLESPACE p_signal_bulk_4,
   PARTITION p_hcni50_bulk_5
     VALUES LESS THAN (96516)
     TABLESPACE p_signal_bulk_5
   PARTITION p_hcni50_bulk_6
     VALUES LESS THAN (97182)
     TABLESPACE p_signal_bulk_6,
   PARTITION p_hcni50_bulk_7
     VALUES LESS THAN (97915)
     TABLESPACE p_signal_bulk_7,
   PARTITION p_hcni50_bulk_8
     VALUES LESS THAN (98504)
     TABLESPACE p_signal_bulk_8,
   PARTITION p_hcni50_bulk_9
     VALUES LESS THAN (99304)
     TABLESPACE p_signal_bulk_9,
   PARTITION p_hcni50_bulk_10
     VALUES LESS THAN (MAXVALUE)
     TABLESPACE p_signal_bulk_10
 ) AS (
   SELECT *
   FROM hcni50_bulk
   WHERE 1 = 0
);
```

Abbildung 6.11: Vorgehen beim Partitionieren der Tabelle ”hcni50_bulk” (3)

Jede Partition wurde dabei in einem separaten Tablespace abgelegt. Ein Tablespace entspricht einer logischen Speichereinheit innhalb der Oracle®-Datenbank, dem ein oder mehrere physikalische Speichereinheiten in Form von Betriebssystemdateien zugeordnet sind. Dieses Vorgehen erhöht die Verfügbarkeit der Tabellenpartitionen und damit der Signaltabellen. Kann z.B. auf eine der Betriebssystemdateien (und damit auf einen Tablespace bzw. eine Partition) nicht zugegriffen werden, können die übrigen Partitionen trotzdem weiter verwendet werden. In Abbildung 6.12 wird ein Auszug der im Data Dictionary vorhandenen Informationen zu den Partitionen der Tabelle ”p_hcni50_bulk” dargestellt. Die Größen der Partitionen sind auf Grund der vorherigen Berechnung der Schussbereiche nahezu identisch.

```
SELECT table_name, partition_name, partition_position AS "POS",
  2  high_value, num_rows, blocks/1024*8 AS "SIZE (MB)"
  3  FROM user_tab_partitions
  4  WHERE table_name = 'P_HCNI50_BULK'
  5* ORDER BY pos

TABLE_NAME     PARTITION_NAME        POS HIGH_VALUE NUM_ROWS SIZE (MB)
-------------- -------------------- --- ---------- -------- ---------
P_HCNI50_BULK  P_HCNI50_BULK_1         1 90918      34974735 963,4141
P_HCNI50_BULK  P_HCNI50_BULK_2         2 94721      34652374 955,4453
P_HCNI50_BULK  P_HCNI50_BULK_3         3 95390      35088564 963,4141
P_HCNI50_BULK  P_HCNI50_BULK_4         4 96154      35154172 963,4141
P_HCNI50_BULK  P_HCNI50_BULK_5         5 96834      34780580 963,4141
P_HCNI50_BULK  P_HCNI50_BULK_6         6 97710      34816278 963,4141
P_HCNI50_BULK  P_HCNI50_BULK_7         7 98371      34957634 963,4141
P_HCNI50_BULK  P_HCNI50_BULK_8         8 99197      34563140 955,4453
P_HCNI50_BULK  P_HCNI50_BULK_9         9 104137     35121444 963,4141
P_HCNI50_BULK  P_HCNI50_BULK_MAXVAL   10 MAXVALUE   34843537 963,4141
```

Abbildung 6.12: Abfragen von Informationen zu Tabellenpartitionen aus dem Data Dictionary

6.3.3 Untersuchte Abfragen

Die Auswirkungen des Partitioning wurden an Hand typisch auftretender Datenbankabfragen bei der Auswertung von TEXTOR-Signalen untersucht. Dabei wurden die Abfragen jeweils unter Verwendung der unpartitionierten und der partitionierten Tabellen durchgeführt.

Insgesamt wurden fünf verschiedene Abfragen durchgeführt. Bei den ersten beiden Abfragen handelt es sich um einfache Abfragen einer Signaltabelle, die einem Zugriff über das TWU entsprechen. Mit den restlichen drei Abfragen wird die Verwendung des TPD Shot Selector simuliert. Abfrage drei und vier entsprechen dabei der Schusssuche über Merkmale 1-dimensionaler Signale. Die fünfte Abfrage spiegelt eine Cutoff-Suche wieder. Tabelle 6.2 zeigt die Einordnung und eine kurze Beschreibung der verschiedenen Abfragen.

Abfrage Id	**Kategorie**	**Beschreibung**
1	TWU	einfache Datenabfrage einer aktuellen Signalversion zu einer vorgegebenen Schussnummer
2	TWU	Datenabfrage aktueller Signalversionen zu einem vorgegebenen Schussbereich mit einer Nebenbedingung
3	TPD Shot Selector	Schusssuche über Merkmale von zwei 1-dimensionalen Signalen innerhalb eines vorgegebenen Schussbereichs
4	TPD Shot Selector	Schusssuche über Merkmale von fünf 1-dimensionalen Signalen innerhalb eines vorgegebenen Schussbereichs
5	TPD Shot Selector	Cutoff-Suche für eine bestimmte Frequenz mit drei Nebenbedingungen innerhalb eines vorgegebenen Schussbereichs

Tabelle 6.2: Beschreibung durchgeführter Datenbankabfragen

Bei **Abfrage 1** werden zur Simulation eines Zugriffs über das TWU die Daten eines Signals zu der aktuellen Version für einen bestimmten Schuss abgefragt. Abbildung 6.13 zeigt den SQL-Code für die Abfrage der unpartitionierten Signaltabelle.

```
SELECT ordinate
FROM hcni50_bulk
WHERE shtid = 95000 AND version = (
      SELECT max(version) FROM hcni50_description
      WHERE shtid = 95000)
```

Abbildung 6.13: SQL-Code für die Abfrage aller Signalwerte einer Schussversion

Weil lediglich die Daten eines Signals zu einer bestimmte Schussnummer ohne eine Nebenbedingung abgefragt werden, ist eine kurze Abfragezeit zu erwarten.

In **Abfrage 2** soll die Abfragezeit erhöht und ein eventuell vorhandener Unterschied bei der Verwendung der partitionierten Tabelle sichtbar gemacht werden. Hierzu wurde, wie in Abbildung 6.14 zu sehen, der abzufragende Schussbereich auf 50 Schussnummern ausgeweitet und zudem die Einschränkung gemacht, nur Zeilen, in der die Ordinate größer Null ist, abzufragen. Da die Menge abzufragender Daten stark vergrößert wurde, die Schussnummern jedoch in einer Tabellenpartition abgelegt sind, sollte die Abfragezeit für die partitionierte Tabelle spürbar kürzer sein.

```
SELECT a.shtid, version, ordinate
FROM hcni50_bulk a
WHERE a.shtid BETWEEN 95000 AND 95050
AND version = (
      SELECT max(version)
      FROM hcni50_description
      WHERE shtid = a.shtid)
AND ordinate > 0
```

Abbildung 6.14: SQL-Code für die Abfrage bestimmter Signalwerte eines Schussbereichs

Da der TPD Shot Selector die Möglichkeit zur Schusssuche an Hand bestimmter Merkmale bietet und daher ein wichtiges Werkzeug für die Auswertung von TEXTOR-Signalen ist, sind die hierbei durchgeführten Abfragen von besonderem Interessen. Da diese Abfragen von der Java-Anwendung im Hintergrund ausgeführt und verarbeitet werden, wurden einige der Abfragen über ein Datenbank-Überwachungs-Tool herausgefiltert und dann für die Zeitmessung herangezogen.
Der erste Abschnitt ”Search Signals” des TPD Shot Selector wurde nicht näher betrachtet, da die Anzeige einer Liste aller Signale und die Suche nach Verwendung bestimmter Heizsysteme innerhalb weniger Sekunden ausgeführt werden und deshalb unkritisch bezüglich der Abfragezeiten sind. Die Suche nach Schüssen, in denen bestimmte Signale berechnet wurden, kann zwar unter Umständen lange (mehrere Minuten) dauern, jedoch liest die im Hintergrund ausgeführte Datenabfrage lediglich die komplette Tabelle ”TPD_STORE” aus. Weil diese Datenbankabfrage sehr schnell durchgeführt wird und bei der Tabelle ”TPD_STORE” kein Partitioning vorgenommen wurde, kann hier ebenfalls auf eine Performance-Betrachtung verzichtet werden.

In **Abfrage 3** wurde eine Schusssuche nach bestimmten Merkmalen 1-dimensionaler Signale, die sich im Abschnitt ”Search with Conditions on one-dimensional signal(s)”

des TPD Shot Selectors durchführen lässt, betrachtet. Die Abfrage greift auf zwei Signaltabellen zu und nutzt einen Schussbereich, der mehrere Partitionen umfasst. Zusätzlich wird der Wertebereich durch eine Bedingung eingeschränkt. Aus den Daten werden jeweils Durchschnittswerte gebildet. Wegen der Verarbeitung der Werte innerhalb der Abfrage und der zahlreichen Tabellenzugriffe wird eine Verbesserung der Abfragezeit durch das verwendete Partitioning erwartet. Die Abbildungen 6.15 und 6.16 zeigen den SQL-Code der im Hintergrund ausgeführten Datenbankabfrage zeigen sowie die Eingabe der Bedingungen über die Weboberfläche.

```
select q1.shtid, q1.mina, q1.maxa, q1.y, q2.y
from (
     select shtid, round(min(abscissa),3) as mina,
            round(max(abscissa),3) as maxa,
            round(AVG(ordinate),2) as y
     from BETAN_BULK a
     where a.shtid between 94000 and 98000
     and  a.version in (
         select max(b.version)
         from BETAN_description b
         where b.shtid=a.shtid
     )
     and a.ordinate between 0 and 0.2
     group by a.shtid
     order by a.shtid desc
     ) q1,
     (
     select shtid, round(min(abscissa),3) as mina,
            round(max(abscissa),3) as maxa,
            round(AVG(ordinate),2) as y
     from BETAP_BULK a
     where a.shtid between 94000 and 98000
     and  a.version in (
         select max(b.version)
         from BETAP_description b
         where b.shtid=a.shtid
     )
     and a.ordinate between 0 and 0.5
     group by a.shtid
     order by a.shtid desc
     ) q2
where q1.shtid=q2.shtid
```

Abbildung 6.15: SQL-Code für die Suche nach Merkmalen 1-dim. Signale

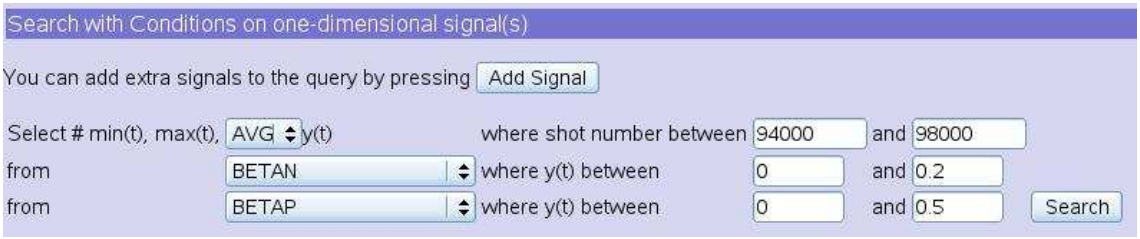


Abbildung 6.16: Schusssuche nach Merkmalen 1-dim. Signale im TPD Shot Selector

In **Abfrage 4** wurden zusätzliche Bedingungen hinzugefügt, dabei aber der zu betrachtende Schussbereich verkleinert. Dadurch soll die in Abfrage 3 erwartete Verbesserung der Abfragezeit verstärkt werden, da nun der Schussbereich deutlich kleiner ist und somit die Anzahl zu berücksichtigender Partitionen reduziert wird. Durch das Einbeziehen weiterer Signaltabellen wird zusätzlich die Komplexität der Abfrage erhöht, was den Zeitunterschied ebenfalls erhöhen könnte. Abbildung 6.17 zeigt die Eingabe der Bedingungen über die Weboberfläche des TPD Shot Selector. Die SQL-Abfrage ist analog zur vorherigen Abfrage aufgebaut, mit dem Unterschied, dass diese nun fünf statt zwei Unterabfragen enthält.

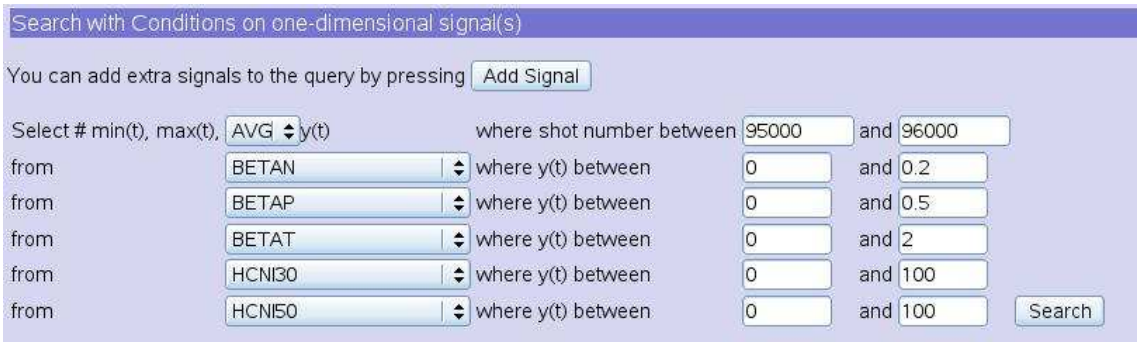


Abbildung 6.17: Erweiterte Suche nach Merkmalen 1-dim. Signale im TPD Shot Selector

Die Ausführung einer Suche über die Merkmale 2-dimensionaler Signale ist sehr komplex aufgebaut, so dass die einzelnen ausgeführten SQL-Statements nur schwer herauszufiltern sind. Zudem hätte eine Zeitmessung der einzelnen SQL-Abfragen wenig Aussagekraft über die innerhalb der Java-Anwendung durchgeführte Suche. Deshalb wurde auf eine Zeitmessung in diesem Abschnitt des TPD Shot Selectors verzichtet.
Stattdessen wurde in **Abfrage 5** eine Cuttoff-Suche simuliert und untersucht. Zwar werden auch bei der Cutoff-Suche verschiedene SQL-Abfragen nacheinander ausgeführt, je-

doch ist die Auswertung und Verarbeitung der Abfragen durch die Java-Anwendung hierbei nicht sonderlich komplex, so dass die Zeitmessung für die nacheinander ausgeführten SQL-Abfragen einen guten Anhaltspunkt für die Gesamtzeit der Suche bietet. Abbildung 6.18 zeigt einen Auszug des SQL-Codes für die durchgeführte Abfrage.

```
select distinct c.shtid
from cutoff105_BULK c
where c.shtid between 102000 and 105000
and c.version in (
    select max(d.version)
    from cutoff105_description d
    where d.shtid=c.shtid );

select distinct a.shtid
from BETAN_BULK a
where a.shtid in ('104090', '104951', ... , '104836' )
and a.version in (
  select max(b.version)
  from BETAN_description b
  where b.shtid=a.shtid )
group by a.shtid
having MIN(a.ordinate) < -1;

select distinct a.shtid
from BETAN_BULK a
where a.shtid in ('104103', '102639', ..., '104108' )
and a.version in (
  select max(b.version)
  from BETAN_description b
  where b.shtid=a.shtid )
group by a.shtid having AVG(a.ordinate) > 0.2;

select distinct a.shtid
from BETAN_BULK a
where a.shtid in ('104103', '102639', ..., '104807' )
and a.version in (
  select max(b.version)
  from BETAN_description b
  where b.shtid=a.shtid )
group by a.shtid having MAX(a.ordinate) > 5;
```

Abbildung 6.18: SQL-Code für eine Cutoff-Suche

Bei der durchgeführten Suche wurden drei Nebenbedingungen eingefügt, um die Komplexität der Abfrage zu erhöhen und den möglichen Vorteil des Partitioning durch weitere Abfragen partitionierter Signaltabellen zu verstärken. Auf Grund des relativ kleinen Schussbereichs wird hier ebenfalls eine Verbesserung der Abfragezeit durch das Partitioning erwartet. Abbildung 6.19 zeigt die Eingabe der Bedingungen für die Cutoff-Suche über die Weboberfläche.

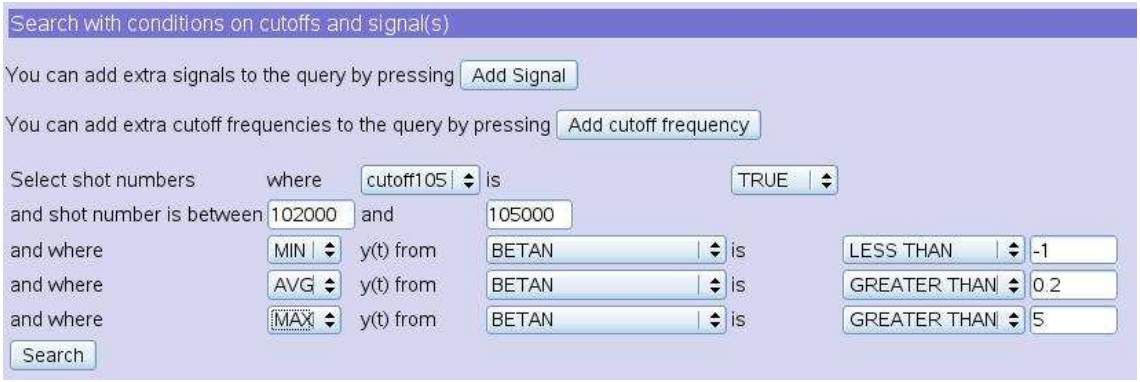


Abbildung 6.19: Cutoff-Suche über die Weboberfläche des TPD Shot Selector

6.3.4 Ergebnisse der Zeitmessungen

Abbildung 6.20 zeigt die Ergebnisse der Zeitmessungen. Zu jeder Abfrage werden die Abfragezeiten (in Sekunden) bei der Verwendung der unpartitionierten (blau) und der partitionierten Signaltabellen (rot) gegenübergestellt.

Gut zu erkennen ist, dass die Zeit für die Durchführung von **Abfrage 1**, dem simulierten Zugriff über das TWU, verschwindend gering im Vergleich zu den weiteren Abfragen ist. Dies liegt daran, dass in Abfrage 1 die abzufragende Datenmenge sehr gering ist und sowohl bei der unpartitionierten als auch bei der partitionierten Tabelle der Zugriff auf die Daten durch Verwendung des Indexes auf die Schussnummernspalte (shtid) beschleunigt werden kann. Die Zeiten für die Abfrage der unpartitionierten und der partitionierten Tabelle sind mit 0,64 Sekunden bzw. 0,63 Sekunden fast identisch. Auf Grund der sehr geringen Abfragezeiten ermöglicht diese Form des Datenzugriffs ein flüssiges Arbeiten mit den Daten und bedarf keiner weiteren Optimierung.

Abfrage 2 dauert mit 33,46 Sekunden bzw. 29,30 Sekunden deutlich länger. Das liegt daran, dass im Vergleich zu Abfrage 1 der Schussbereich und damit die Menge abzufra-

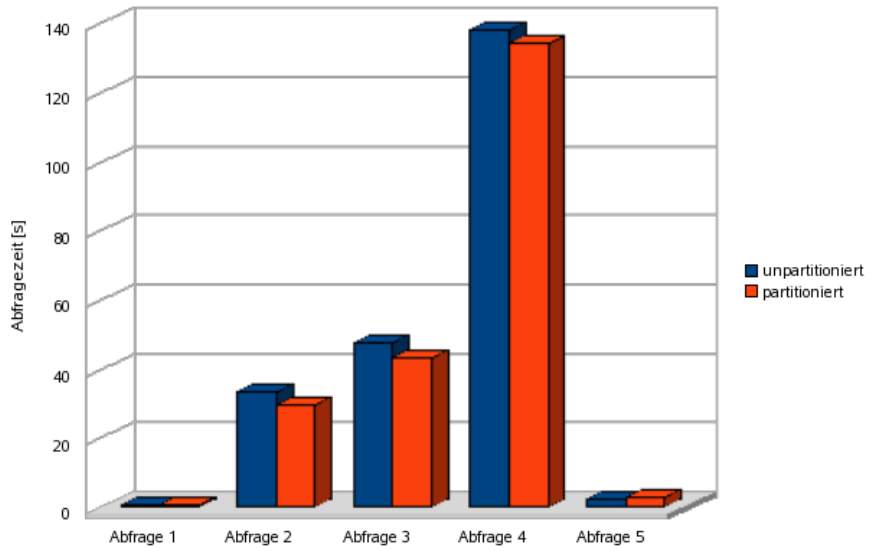


Abbildung 6.20: Abfragezeiten bei (un-)partitionierten Tabellen

gender Daten erheblich höher ist. Entgegen den Erwartungen ist die Beschleunigung der Datenbankabfrage durch die Verwendung partitionierter Signaltabellen nur sehr gering. Um dies genauer zu analysieren, wurde die Abfrage noch einmal ohne das Auslesen der Daten durchgeführt. Stattdessen wurde nur die Anzahl selektierter Zeilen gezählt. Ändert man die Abfrage so ab, dass statt der Daten nur die Anzahl der ausgewählten Zeilen abgefragt wird (Verwendung von ”count (*)”), betragen die Abfragezeiten mit 1,64 Sekunden und 1,82 Sekunden nur noch einen Bruchteil der zuvor gemessenen Zeiten. Die meiste Zeit wird also offensichtlich für das Auslesen der Daten benötigt. Die große Anzahl auszulesender Datensätze von ca. 2,8 Millionen ist bei dieser Abfrage verantwortlich für die lange Abfragezeit. Da der Vorgang des Datenlesens durch das Partitioning nicht beschleunigt werden kann, führt die Verwendung der partitionierten Tabellen nur zu einem geringen Zeitvorteil.

Bei der simulierten Suche über den TPD-Shot Selector in **Abfrage 3** sind die Vorteile des Partitioning ebenfalls gering (42,98 Sekunden gegenüber 47,23 Sekunden). Wie in der vorigen Abfrage wurden daher die Abfragen ohne Auslesen der eigentlichen Werte (also wieder mit ”count(*)”) ausgeführt. Weil die Abfragezeiten jedoch nahezu identisch blieben, wurden zusätzlich die Ausführungspläne der Abfrage für die unpartitionierten und die partitonierten Tabellen verglichen. Abbildung 6.21 zeigt den Ausführungsplan bei der Verwendung der unpartitionierten Tabellen.

```
-------------------------------------------------------------------
| Id  | Operation                          | Name                 |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                      |
|*  1 |  HASH JOIN                         |                      |
|   2 |   VIEW                             |                      |
|   3 |    SORT GROUP BY                   |                      |
|*  4 |     FILTER                         |                      |
|*  5 |      TABLE ACCESS FULL             | BETAN_BULK           |
|   6 |      SORT AGGREGATE                |                      |
|   7 |       TABLE ACCESS BY INDEX ROWID| BETAN_DESCRIPTION    |
|*  8 |        INDEX RANGE SCAN            | BETAN_DESCRIPTION_IDX|
|   9 |   VIEW                             |                      |
|  10 |    SORT GROUP BY                   |                      |
|  11 |     VIEW                           |                      |
|* 12 |      FILTER                        |                      |
|  13 |       HASH GROUP BY                |                      |
|* 14 |        HASH JOIN                   |                      |
|* 15 |         TABLE ACCESS FULL          | BETAP_BULK           |
|* 16 |         TABLE ACCESS FULL          | BETAP_DESCRIPTION    |
-------------------------------------------------------------------
```

Abbildung 6.21: Ausführungsplan für die Abfrage der unpartitonierten Tabellen

Wie zu erkennen ist, wird die Verknüpfung der Tabellen über eine JOIN-Operation von zwei intern erstellten Views durchgeführt. Die Daten der Signaltabellen werden über zwei ”full-table-scans” eingelesen. Das heißt, es werden die kompletten Tabellen statt ausgewählter Zeilen eingelesen. Die Einschränkung auf diejenigen Zeilen, die relevant für die Abfrage sind (also die Kriterien der WHERE-Bedingungen in der Abfrage erfüllen) werden anschließend durch eine FILTER-Operation extrahiert. Für das Auslesen der Beschreibungstabelle BETAN_DESCRIPTION wird der zu Grunde liegende Index (Operation 8) benutzt, während die Beschreibungstabelle für das Signal BETAP komplett ausgelesen wird (Operation 16). Die Operation ”SORT GROUP BY” (3 und 10) wird für die Berechnung der Minimum-, Maximum- und Durchschnittswerte benötigt.

Der in Abbildung 6.22 zu sehende Ausführungsplan bei der Verwendung von partionierten Tabellen unterscheidet sich nur wenig. Der Zugriff auf die beiden Signaltabellen P_BETAN_BULK und P_BETAP_BULK erfolgt hier unter Ausnutzung der Partitionierung. Statt der gesamten Tabellen werden nur die relevanten Partitionen ausgelesen. Der

verwendete Partitionsbereich wird in den Spalten ”Pstart” und ”Pstop” angezeigt. Diese Tabellenzugriffe erklären den leichten Zeitvorteil (42,98 Sekunden gegenüber 47,24 Sekunden) bei der Verwendung der partitionierten Tabellen. Weil die übrigen Operationen bei der Ausführung jedoch identisch sind, wird zugleich klar, warum die der Abfragezeit nicht deutlicher ist.

```
-------------------------------------------------------------------------------
| Id  | Operation                      | Name                  | Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                       |       |       |
|*  1 |  HASH JOIN                     |                       |       |       |
|   2 |   VIEW                         |                       |       |       |
|   3 |    SORT GROUP BY               |                       |       |       |
|*  4 |     FILTER                     |                       |       |       |
|   5 |      PARTITION RANGE ITERATOR  |                       |     5 |     1 |
|*  6 |       TABLE ACCESS FULL        | P_BETAN_BULK          |     5 |     1 |
|   7 |      SORT AGGREGATE            |                       |       |       |
|   8 |       TABLE ACCESS BY INDEX ROWID| BETAN_DESCRIPTION     |       |       |
|*  9 |        INDEX RANGE SCAN        | BETAN_DESCRIPTION_IDX |       |       |
|  10 |   VIEW                         |                       |       |       |
|  11 |    SORT GROUP BY               |                       |       |       |
|  12 |     VIEW                       |                       |       |       |
|* 13 |      FILTER                    |                       |       |       |
|  14 |       HASH GROUP BY            |                       |       |       |
|* 15 |        HASH JOIN               |                       |       |       |
|  16 |         PARTITION RANGE ITERATOR |                     |     1 |     4 |
|* 17 |          TABLE ACCESS FULL     | P_BETAP_BULK          |     1 |     4 |
|* 18 |         TABLE ACCESS FULL      | BETAP_DESCRIPTION     |       |       |
-------------------------------------------------------------------------------
```

Abbildung 6.22: Ausführungsplan für die Abfrage der partitonierten Tabellen

Obwohl der Schussbereich bei **Abfrage 4** so gewählt wurde, dass jeweils nur 2 Partitionen der Signaltabellen berücksichtigt werden müssen, wird auch bei der zweiten Suche nach Merkmalen 1-dimensionaler Signale nur ein kleiner Geschwindigkeitsvorteil durch die Verwendung der partitionierten Tabellen erreicht. Der in Abfrage 3 auftretende leichte Zeitvorteil des Partitioning konnte also durch Einschränkung des Schussbereichs und Hinzufügen weiterer Nebenbedingungen (und Signaltabellen) nicht verstärkt werden. Dies kann wiederum durch die geringe Auswirkung des Partitioning auf den Ausführungsplan der Abfrage begründet werden. Wegen der vielen Bedingungen bei der Suche dauert die Abfrage mit 137,81 Sekunden bzw. 134,27 Sekunden sehr lange.

Bei der simulierten Cutoff-Suche in **Abfrage 5** bringt das Partitioning wie zuvor nur einen kleinen Vorteil bezüglich der Abfragezeit. Auffällig ist, dass trotz zusätzlicher Nebenbedingungen und des relativ großen Schussbereichs die Abfragezeit z.T. erheblich geringer ist als bei der Suche an Hand von Merkmalen 1-dimensionaler Signale. Eine Erklärung

hierfür ist, dass die Durchführung der Cutoff-Suche bereits bei der Implementierung innerhalb der Java-Anwendung optimiert wurde. Durch die Verwendung aufeinanderfolgender wenig komplexer SQL-Abfragen mit jeweils durch die vorigen Abfragen eingeschränkten Schussmengen kann bei der Cutoff-Suche die Abfragezeit gering gehalten werden. Nähere Informationen zur Vorgehensweise bei der Cutoff-Suche sind in [LIT16] zu finden.

Insgesamt bewirkt die Verwendung partitionierter Tabellen leider nur eine sehr geringe Verbesserung der Abfragezeiten. Dies liegt hauptsächlich an dem großen Einfluss des Auslesens größerer Datenmengen auf die Abfragezeit. Da das Partitioning den Vorgang des Datenlesens nicht beschleunigen kann, können die Abfragezeiten deshalb durch Verwendung partitionierter Tabellen nur unbedeutend verbessert werden.
Die Vorgehensweise im TPD Shot Selector, die Suche innerhalb der Java-Anwendung bereits in mehrere Abfragen zu zerlegen und innerhalb der Anwendung zu verknüpfen trägt ebenfalls dazu bei, dass das Partitioning nur wenig Auswirkung zeigt.

6.4 Einsatz von Materialized Views

Ein weiterer Ansatz für die Beschleunigung der Signalabfragen ist die Verwendung von Materialized Views. Deren Eigenschaften werden zunächst im Allgemeinen beschrieben, bevor der mögliche Einsatz in der TPD und die Umsetzung innerhalb der Testumgebung erläutert werden. Anschließend werden die Ergebnisse der Zeitmessungen präsentiert und bewertet.

6.4.1 Eigenschaften von Materialized Views

Innerhalb einer Datenbank stellen Views bestimmte Ansichten für die Daten ein oder mehrerer Tabellen bereit. Dabei werden die Informationen, die durch einen View angezeigt werden, durch den Zugriff auf die Basistabellen über die Ausführung entsprechender SELECT-Statements bereitgestellt. Bei jedem Zugriff auf einen View müssen diese Abfragen jedoch neu ausgewertet werden.
Bei Materialized Views (oder auch ”Summary Tables” genannt) werden dagegen die Daten der Ansicht gespeichert, so dass bei einem Zugriff auf Materialized Views keine Abfragen der Basistabellen ausgewertet werden müssen. Wie bei eigenständigen Tabellen ist eine Indizierung und Partitionierung möglich.
Ein Vorteil von Materialized Views ist, dass die Aktualisierung automatisch bei einer Änderung der zu Grunde liegenden Basistabellen erfolgen kann. Zudem kann bei Abfragen der Basistabellen ggf. (durch ein sogenanntes ”Query Rewrite”) auf die Materialized Views zugegriffen und dadurch eine Beschleunigung der Abfragen erzielt werden. Dies geschieht automatisch und für die Anwendung / den Anwender transparent.

6.4.2 Einsatz von Materialized Views in der TPD

Weil das Auslesen großer Datenmengen viel Zeit in Anspruch nimmt, sind für die Abfragezeiten von Signaldaten die großen Datenmengen, die beim Herausfiltern der relevanten Informationen zu berücksichtigen sind, problematisch. Bei der Auswertung von Signaldaten und der Abfrage über das TWU bzw. den TPD Shot Selector werden lediglich die aktuellen Versionen eines Signals berücksichtigt. Alle vorherigen Signalversionen, die ebenfalls in den Signaltabellen gespeichert sind und die Abfragekriterien (z.B. bezüglich der Schussnummer) erfüllen, müssen herausgefiltert werden. Diese Einschränkung auf die aktuellen Signalversionen wird mit Hilfe der in Abbildung 6.23 dargestellten Unterabfrage durchgeführt.

```
... WHERE a.version IN (
         SELECT max(b.version)
         FROM ..._description b
         WHERE b.shtid=a.shtid
         )
```

Abbildung 6.23: Unterabfrage zur Einschränkung auf die aktuellen Signalversionen

Durch die Verwendung von Materialized Views, die jeweils nur eine aktuelle Version der Signale enthalten, kann auf das Filtern der Signalversionen und die oben dargestellte Unterabfrage verzichtet werden. Auf diese Weise kann die Komplexität der Abfragen reduziert und evtl. die Abfragezeit verbessert werden. Für entsprechende Zeitmessungen wurden daher in der Testumgebung 5 Materialized Views inklusive zugehöriger Indizes über die Schussnummer erstellt. Da die Basistabellen in der Testumgebung statisch sind, wurde hier keine automatische Aktualisierung der Materialized Views eingerichtet. Abbildung 6.24 zeigt ein SQL-Beispiel für das Erzeugen der Materialized View ”MV_BETAN_BULK”.

Durch das Reduzieren der Datensätze auf diejenigen mit aktuellen Versionsnummern sind die Materialized Views im Vergleich zur jeweiligen Basistabelle deutlich kleiner. Dies hängt jedoch von der jeweiligen Signaltabelle und der Anzahl verschiedener Versionen zu den einzelnen Schüssen ab. Wurde ein Signal in der Regel nur einmal pro Schuss berechnet, sind die Signaltabelle und die daraus erstellte Materialized View folglich ungefähr gleich groß. Bei Signalen, die häufig neu berechnet werden, kann dagegen der Größenunterschied durchaus sehr deutlich sein. Abbildung 6.3 zeigt die Anzahl der Zeilen und die Größe der Basistabellen und der zugehörigen Materialized Views. Wie zu erkennen

```
CREATE MATERIALIZED VIEW mv_betan_bulk
TABLESPACE ipp_prod
REFRESH COMPLETE ON DEMAND
  SELECT * FROM betan_bulk a
  WHERE a.version = (
    SELECT max(b.version)
    FROM betan_bulk b
    WHERE b.shtid = a.shtid
  )
```

Abbildung 6.24: SQL-Code für das Erstellen einer Materialized View

ist, belegen die für die Zeitmessungen erstellten Materialized Views nahezu gleich viel, die letzten beiden sogar etwas mehr Speicherplatz als die Basistabellen. Die Zeilenanzahl konnte hingegen teilweise deutlich reduziert werden.

Objektname	**Objekttyp**	**Zeilenanzahl**	**Größe (MB)**
BETAN_BULK	Tabelle	34732232	819,9766
MV_BETAN_BULK	Materialized View	29237682	766,8672
BETAP_BULK	Tabelle	33824307	796,0703
MV_BETAP_BULK	Materialized View	28546369	743,6719
BETAT_BULK	Tabelle	35075948	827,9453
MV_BETAT_BULK	Materialized View	29521576	776,875
HCNI30_BULK	Tabelle	4,68E+08	11739,73
MV_HCNI30_BULK	Materialized View	4,56E+08	12678,43
HCNI50_BULK	Tabelle	3,49E+08	8627,82
MV_HCNI50_BULK	Materialized View	3,42E+08	9426,008

Tabelle 6.3: Größenvergleich zwischen Basistabellen und Matrialized Views

6.4.3 Ergebnisse der Zeitmessungen

Für die Zeitmessungen bei der Verwendung von Materialized Views wurden dieselben Abfragen wie zuvor bei der Untersuchung des Partitionings verwendet, da diese sowohl den Zugriff über das TWU und den TPD Shot Selector als auch die Auswertung unterschiedlich komplexer SQL-Statements abdecken. Allerdings konnten durch die Verwen-

dung der zuvor erstellten Materialized Views die Unterabfragen für das Herausfiltern der aktuellen Signalversionen (s. Abbildung 6.23) entfernt werden. Besonders bei der Schussnummernsuche über die Merkmale mehrerer 1-dimensionaler Signale wird eine Verbesserung der Abfragezeit erwartet, da hier gleich mehrere Unterabfragen entfallen konnten. Abbildung 6.25 zeigt die Ergebnisse der Zeitmessungen.

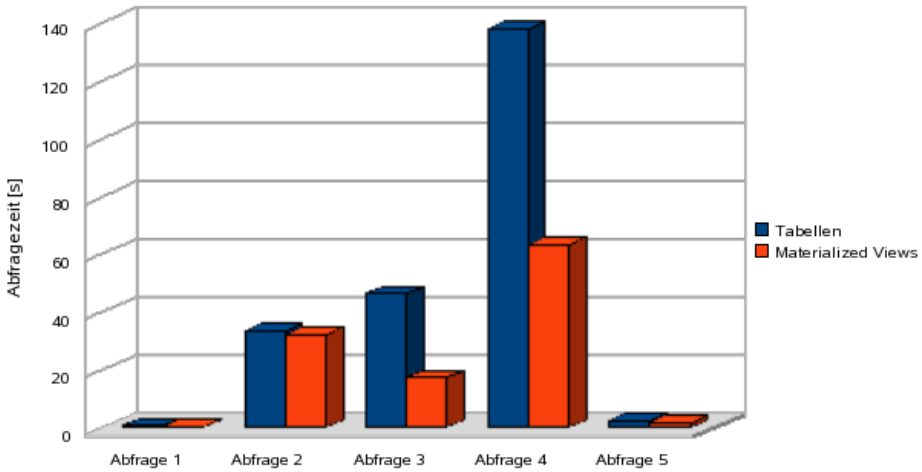


Abbildung 6.25: Abfragezeiten bei Verwendung von Tabellen und Materialized Views

Für **Abfrage 1** und **Abfrage 2** wird keine spürbare Verbesserung der Abfragezeiten erreicht. Dies ist jedoch wenig verwunderlich, da in der ersten Abfrage lediglich ein Schuss untersucht werden muss und das Filtern der aktuellen Version hierfür keinen spürbaren Mehraufwand bedeutet. In der zweiten Abfrage ist, wie beim Partitioning bereits festgestellt, das Auslesen der vielen Daten ausschlaggebend für die Abfragezeit. Auch hier fällt das Filtern der Versionsnummern bei nur 50 Schussnummern nicht ins Gewicht, so dass sich die Abfragezeiten mit 33,45 Sekunden und 32,30 Sekunden nur geringfügig unterscheiden.

Wie erwartet ist bei den **Abfragen 3** und **4** eine deutliche Verbesserung der Abfragezeiten zu erkennen. Mit 47,47 Sekunden gegenüber 17,28 Sekunden bzw. 137,81 Sekunden gegenüber 63,45 dauern die Abfragen bei Verwendung der Materialized Views nichct halb so lang. Dies liegt vor allem an der zum Teil stark reduzierten Anzahl zu berücksichtigender Datensätze sowie den fehlenden Unterabfragen.

Bei der simulierten Cutoff-Suche in **Abfrage 5** kann ebenfalls ein Zeitvorteil erreicht werden. Da die Abfrage jedoch, wie in Kapitel 6.3.4 erwähnt, bereits bei der Implementierung der Cutoff-Suche optimiert und die Menge herauszufilternder Signalversionen reduziert wurde, ist hier der Zeitunterschied mit 2,22 Sekunden und 1,7 Sekunden eher gering.

Insgesamt ist die Verwendung von Materialized Views mit der Beschränkung auf die aktuellen Versionen der Signale von Vorteil und kann die Signalabfragen spürbar beschleunigen. Dies hängt jedoch von der jeweiligen Abfrage und den Eigenschaften der zu Grunde liegenden Basistabellen ab (z.B. ob viele verschiedene Versionen enthalten sind). Zudem ist zu berücksichtigen, dass die Materialized Views zusätzlich zu den bereits sehr großen Signaltabellen weiteren Speicherplatz belegen, der unter Umständen der Basistabelle mit seiner Größe sehr nahe kommt.
Für einen Einsatz in der Produktionsumgebung ist darüber hinaus eine automatische Aktualisierung der Materialized Views nötig, damit keine veralteten Daten angezeigt werden. Da ein Materialized View während der Aktualisierung (”Refresh”) in der Regel nicht abgefragt werden kann, bedeutet dies unter Umständen eine zeitliche Einschränkung für die Durchführung von Suchabfragen.

6.5 Verwendung zusätzlicher Indizes

Indizes in der Datenbank stellen einen Zugriff auf die Datensätze über Zeiger bereit. Dadurch können die Datensätze schneller gefunden und die Datenbankabfragen besonders bei großen Tabellen unter Umständen stark beschleunigt werden. In der TPD werden bereits Indizes über die Schussnummern der Signal- und Beschreibungstabellen eingesetzt. Es soll nun untersucht werden, ob eine Erweiterung der Indizierung zur Verbesserung der Abfragezeiten beitragen kann.

6.5.1 Vorgehen bei der Indexerstellung

Voraussetzung für die Verwendung eines Indexes bei der Ausführung einer Datenbankabfrage ist die Verwendung der Indexspalte(n) in einer WHERE-Bedingung der Abfrage. Zudem kann nur dann ein Vorteil erzielt werden, wenn die Bereichsmenge durch diese Nebenbedingung spürbar eingeschränkt wird. Andernfalls ist es günstiger, die komplette Tabelle einzulesen, statt die gesuchten Datensätze über Index-Zugriffe zu ermitteln.
Ein Beispiel für die Performance-Verbesserung durch Verwendung eines Indexes stellt die bei den vorherigen Zeitmessungen verwendete Abfrage 1 dar, bei der die Daten für das Signal HCNI50 im Schuss 95000 abgefragt werden. Mit Verwendung des Indexes auf die Spalte ”SHTID” dauert die Abfrage lediglich ca. 0.6 Sekunden. Ohne Nutzung des Indexes werden für dieselbe Abfrage ca. 22 Sekunden benötigt !

Da bei den betrachteten Abfragen neben der Schussnummer stets die Versionsnummer auf die der aktuellen Version eingeschränkt wird, bietet sich die Erweiterung der Indizierung auf die Spalte ”VERSION” an. Weil die beiden Spalten ”SHTID” und ”VERSION” bei den Nebenbedingungen der Abfragen in Kombination auftreten, wird ein zusammengesetzter Index verwendet. Ziel dabei ist es, das Herausfiltern der aktuellen Signalversion zu beschleunigen und damit wie beim zuvor untersuchten Einsatz von Materialized Views die Abfragezeiten zu verbessern. Die entsprechenden Indizes können mit dem in Abbildung 6.26 SQL-Statement erstellt werden.

```
CREATE INDEX betan_bulk_idx ON betan_bulk (shtid, version)
```

Abbildung 6.26: Beispiel für das Erstellen eines zusammengesetzten Indexes

Ebenfalls wäre eine Indizierung der Spalte ”ORDINATE” für die Signaltabellen der TPD denkbar, da diese in einigen der Abfragen als Nebenbedingung verwendet wird. Da es bei den eigentlichen Signalwerten jedoch vorkommen kann, dass ein Wert (z.B. Null) sehr häufig auftaucht oder sich sehr viele Werte in einem kleinen Wertebereich befinden, macht eine Indizierung dieser Spalte nicht viel Sinn und wurde daher nicht näher untersucht.

Für die Zeitmessungen wurden ebenfalls die in Kapitel 6.3.3 dargestellten Abfragen verwendet. Es wird erwartet, dass - wie bei den Materialized Views - besonders bei den Abfragen 3 und 4 eine Verbesserung der Abfragezeiten zu erkennen ist.

6.5.2 Ergebnisse der Zeitmessungen

Wie in Abbildung 6.27 zu erkennen, tritt bei keiner der durchgeführten Abfragen ein deutlicher Zeitunterschied für die Verwendung eines einfachen Indexes (Spalte SHTID) und eines zusammengesetzten Indexes (Spalten SHTID und VERSION) auf. Die Maßnahme der Indexerweiterung blieb im Hinblick auf eine mögliche Performance-Verbesserung also wirkungslos.

Zudem belegen die zusammengesetzten Indizes gegenüber den einfachen Indizes auf die Schussnummer zum Teil deutlich mehr Speicherplatz. Für die Tabelle ”HCNI30_BULK” z.B. beträgt der Unterschied ca. 1,7 GB.

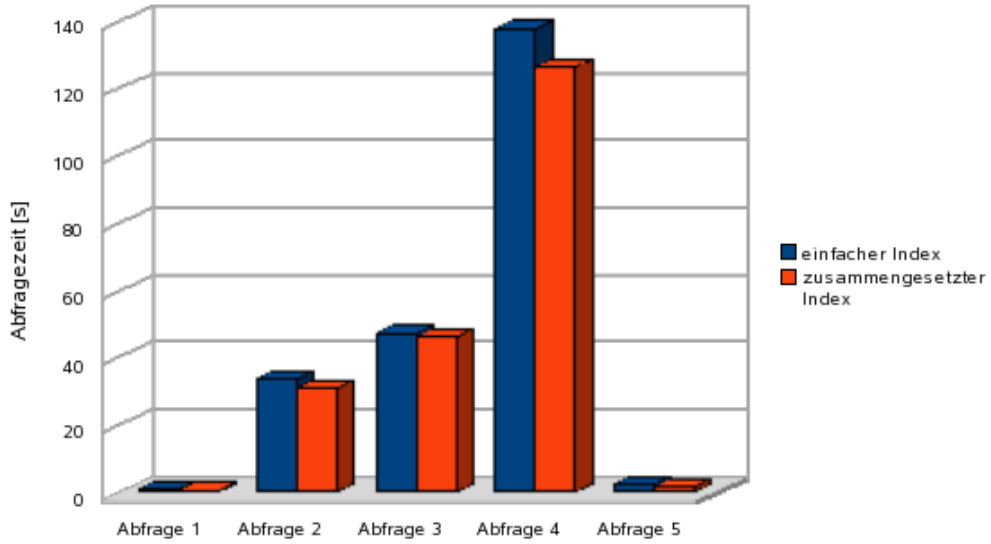


Abbildung 6.27: Abfragezeiten bei Verwendung verschiedener Indizes

6.6 Weboberfläche des TPD Shot Selector

Da die Benutzer in der Regel keine direkten Datenbankabfragen ausführen, sondern die Abfrage von Signaldaten über das TWU oder den TPD Shot Selector durchführen, sind neben der für die reinen Datenbankabfragen benötigten Zeiten auch die Zeiten vom Absenden der Anfrage des Clients bis zur Anzeige der Ergebnisse von Interesse.
Während die Abfragen von Signaldaten über das TWU in der Regel wenig komplex sind und ein flüssiges Arbeiten ermöglichen, werden beim TPD Shot Selector über die Weboberfläche und das dahinterliegende Java-Servlet zum Teil sehr komplexe Datenbankabfragen erzeugt, ausgeführt und ausgewertet. Daher wurden zur Weboberfläche des TPD Shot Selector ebenfalls einige Messungen durchgeführt.

6.6.1 Ablauf einer Anfrage über die Weboberfläche

Zunächst muss die Anfrage vom Client des Benutzers zum Servlet des TPD Shot Selector transportiert werden. Dort müssen die Formulareingaben ausgewertet und entsprechende Datenbankabfragen generiert und zum Datenbank-Server geschickt werden.
Die Ergebnisse des Datenbankservers werden anschließend zurück an das Java-Servlet geschickt, wo sie ggf. aufbereitet und als Webseite an den Client des Benutzers gesendet werden. Abbildung 6.28 skizziert den Ablauf einer Anfrage unter Verwendung der We-

boberfläche des TPD Shot Selector.

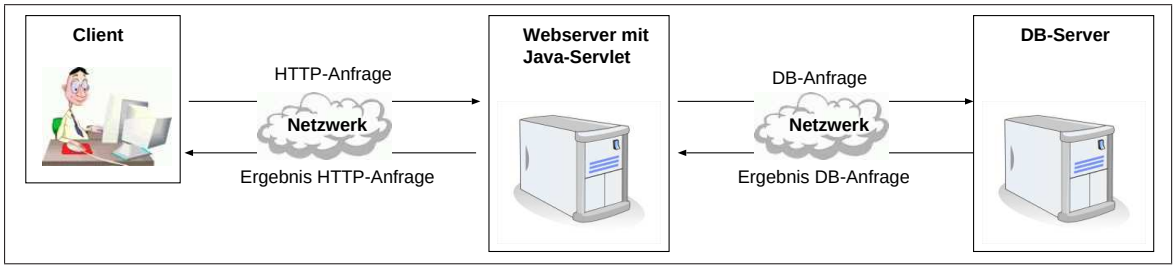


Abbildung 6.28: Ablauf einer Anfrage über die Weboberfläche des TPD Shot Selector

6.6.2 Abfragezeiten bei Vewendung der Weboberfläche

Die Abfragen 3 bis 5 aus den bisherigen Zeitmessungen spiegeln den Zugriff über die Weboberfläche des TPD Shot Selector wieder und können durch entsprechende Eingaben im Webformular erzeugt werden (vgl. Kapitel 6.3.3).Daher wurden diese Abfragen ebenfalls für die Zeitmessungen über die Weboberfläche herangezogen.

Weil eine Zeitmessung innerhalb des Webbrowsers schwierig zu automatisieren ist, wurde hierzu ein weiteres PHP-Skript verwendet. Dieses greift auf die cURL[3] -Bibliothek zu, mit der sich Client-Anfragen an einen Server simulieren und auswerten lassen. Auf diese Weise lassen sich z.B. Webseiten ohne einen Browser abrufen. Mit cURL ist es möglich, beim Aufruf einer URL Parameter zu übergeben. Dadurch lassen sich die Formulareingaben im TPD Shot Selector automatisieren und in das PHP-Skript einbinden. Das Servlet des TPD Shot Selector wertet die mit Hilfe der cURL-Bibliothek aus dem PHP-Skript erzeugte Anfrage wie eine gewöhnliche Browser-Anfrage aus. Die Ergebnisseite steht anschließend in PHP zur Verfügung. Die eigentliche Zeitmessung kann analog zu den vorigen Performanceuntersuchungen mit Hilfe der PHP-Funktion ”microtime” erfolgen.

Wie in Abbildung 6.29 zu erkennen, dauert der Zugriff über die Weboberfläche bei allen 3 Abfragen etwas länger. Diese Zeitunterschiede hängen zum einen mit der Übertragung der Daten über das Netzwerk und zum anderen mit der Generierung und Auswertung der Datenbankabfragen innerhalb des Java-Servlets zusammen.

[3]cURL (client URL): freie Software zur Übertragung von Dateien unter Verwendung von URL-Syntax

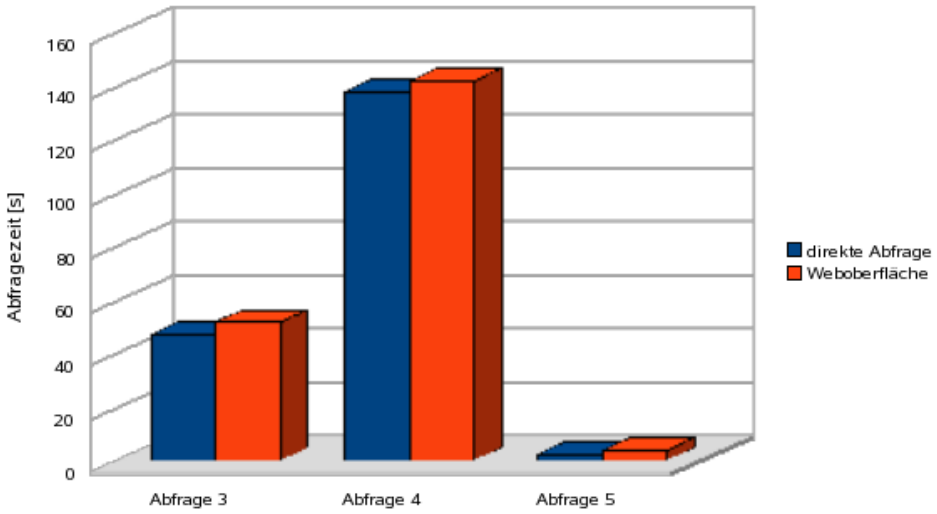


Abbildung 6.29: Vergleich der Abfragezeiten bei direkter Ausführung und unter Verwendung der Weboberfläche

Wie zu sehen ist, verlangsamt die Verwendung der Weboberfläche die Ausführung der Anfragen jedoch nur geringfügig und stellt daher durch die komfortable Eingabe von Parametern über Formularfelder sowie die automatische Erzeugung der Datenbankabfragen ein nützliches Hilfsmittel bei der Auswertung von Textorsignalen dar.

6.7 Vergleich mit der Produktionsumgebung

Die bisherigen Zeitmessungen wurden alle auf der dafür eingerichteten Testumgebung vorgenommen. Da eine Migration der Produktionsumgebung auf den für die Testumgebung verwendeten Server bevorsteht, ist ebenfalls ein Vergleich der Abfragezeiten mit denen des derzeitigen Produktionsservers von Interesse. Daher wurden die 5 Beispielabfragen aus den vorigen Zeitmessungen ebenfalls auf dem Datenbankserver der Produktionsumgebung ausgeführt (ohne Verwendung der Weboberfläche). Abbildung 6.30 zeigt die Ergebnisse der Zeitmessungen.
Wie zu erkennen ist, dauern die Abfragen auf der Produktionsumgebung deutlich länger. Für **Abfrage 4** wird z.B. gegenüber der Testumgebung ungefähr 9-mal soviel Zeit benötigt.

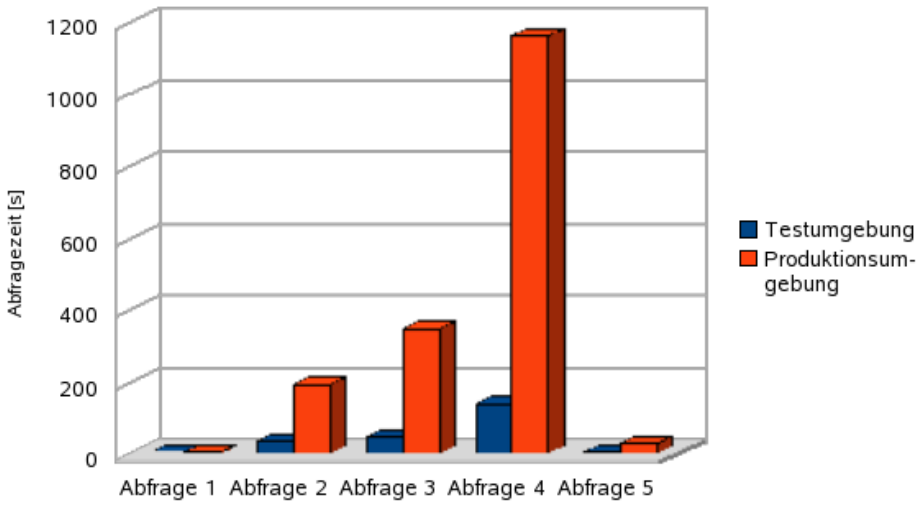


Abbildung 6.30: Vergleich der Abfragezeiten innerhalb der Test- und Produktionsumgebung

Da die Zeitunterschiede so enorm sind, wurde die Hardware der beiden Systeme näher betrachtet. Wie in Tabelle 6.4 zu sehen ist, besitzt der Server der Testumgebung zwar nur zwei Prozessoren, diese enhalten jedoch jeweils vier Rechenkerne und sind deutlich höher getaktet als die vier in der Produktionsumgebung verwendeten CPUs. Zudem enthält der Testserver doppelt soviel Arbeitsspeicher wie der Server der Produktionsumgebung.

	Testumgebung	**Produktionsumgebung**
CPU-Anzahl	2	4
CPU-Typ	Intel XEON® Quadcore (2.33 GHz)	Power 3 (375 Mhz)
Arbeitsspeicher	16 GB	8 GB
Plattenplatz (intern)	140 GB	35 GB
Plattenplatz (extern)	2.1 TB auf Storage (RAID 5)	1.3 TB auf Storage (RAID 1)
Hersteller	DELL	IBM
OS	Oracle Enterprise Linux® 5.3 64 Bit	AIX® 5.2

Tabelle 6.4: Systemvergleich zwischen Produktions- und Testsystem

Abbildung 6.31 zeigt den Verlauf der CPU-Auslastungen sowie der Plattenzugriffe während der Ausführung der Beispielabfragen.

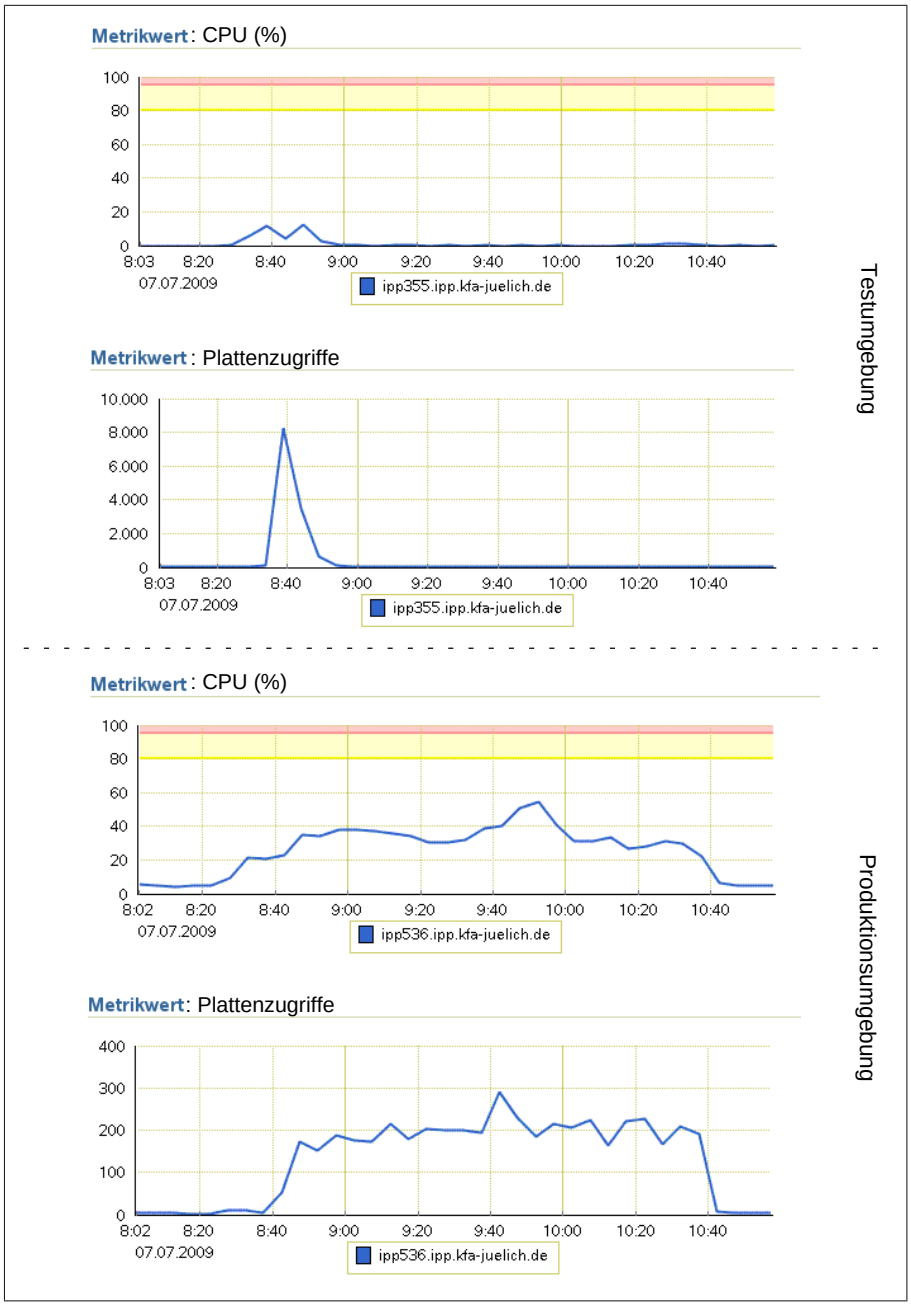


Abbildung 6.31: Systemauslastung während der Durchführung der Abfragen auf dem Test- und Produktionssystem

Während die CPU-Last auf dem Produktionsserver zwischenzeitlich auf über 50% steigt und durchschnittlich bei ca. 40% liegt, werden in der Testumgebung nur ca. 12% CPU-Auslastung erreicht. Zudem dauert die Ausführung hier weniger als eine halbe Stunde, während auf dem Produtkionsserver mehr als zwei Stunden benötigt werden. Die maximale CPU-Auslastung von ca. 12% in der Testumgebung lässt sich dadurch erklären, dass bei der Ausführung der Abfragen nur eine Datenbanksession verwendet wird, in der die Datenbankabfragen der Reihe nach durchgeführt werden. Auf dem Testsystem wird der zur Session gehörige Betriebssystemprozess auf einem der 8 zur Verfügung stehenden CPU-Kerne ausgeführt. Da der einzelne CPU-Kern bis zu 100% ausgelastet wird, ergibt sich für die gesamte CPU-Auslastung ein Wert von ca. 12% ($\sim$ 1/8). Die CPU-Last in der Produktionsumgebung war im Gegensatz zur Testumgebung während der Abfragen relativ gleichmäßig auf die 4 CPUs verteilt.
Auf dem Produktionssystem gibt es einige weitere Prozesse, die spürbar zur Auslastung der CPU beitragen. Neben der Datenbanksoftware wird zusätzlich ein Webserver betrieben, der mit ca. 50 Betriebssystemprozessen ebenfalls Systemlast erzeugt. Deshalb wird auf dem Produktionssystem, wie in der Grafik zu erkennen, vor und nach der Ausführung der Abfragen ca. 5 % CPU-Last erreicht, während die Systemlast in der Testumgebung vor und nach der Abfrage nahezu Null ist.
Die Plattenzugriffe unterscheiden sich ebenfalls deutlich. In der Produktionsumgebung werden nur knapp 300 Ein-/Ausgabeoperationen pro Sekunde erreicht, bei der Testumgebung sind es dagegen über 8000. Da die CPU-Auslastung auf dem Produktionsserver mit ca. 40% nicht ausgereizt ist, liegt die Vermutung nahe, dass hier die Plattenzugriffe die Durchführung der Abfragen bremsen.

6.8 Kapitelzusammenfassung

In diesem Kapitel wurden verschiedene Ansätze zur Performanceverbesserung von Signalabfragen in der TPD untersucht. Hierzu wurden in einer Testumgebung verschiedene Zeitmessungen zu typisch auftretenden Signalabfragen über das TWU und den TPD Shot Selector durchgeführt.
Die Aufteilung großer Signaltabellen mit Hilfe einer Tabellenpartitionierung sowie die Erweiterung der Tabellenindizierung bewirkten keine deutlichen Verbesserungen der Abfragezeiten. Dagegen konnte durch die Verwendung von Materialized Views, die jeweils nur aktuelle Signalversionen enthalten, die Abfragen spürbar beschleunigt werden.
Die Zeitmessungen zeigten, dass die Nutzung der Weboberfläche des TPD Shot Selector im Vergleich zur direkten Ausführung der Abfragen auf dem Datenbankserver nur unwesentlich langsamer ist. Die Durchführung der Abfragen innerhalb der Produktionsumgebung dauert unter anderem auf Grund der Hardwareunterschiede im Vergleich zur Testumgebung erheblich länger.

Kapitel 7

Zusammenfassung

Im Forschungszentrum Jülich wird das zentrale Großgerät TEXTOR (TOKAMAK Experiment for Technology Oriented Research) betrieben, um Erkenntnisse über die Abläufe bei der Kernfusion zu gewinnen. Im Mittelpunkt steht dabei die Erforschung von Plasma-Wand-Wechselwirkungen und Plasmainstabilitäten. Im Rahmen der Arbeit wurde die Datenverarbeitung der Experimentdaten und insbesondere das Erstellen und Abfragen von Signaldaten in der TEXTOR Physical Database (TPD) betrachtet. Neben der Implementierung einer automatischen Neuberechung abhängiger Signale in der TPD wurde die Verbesserung der Abfragezeiten für Signaldaten unter Verwendung verschiedener Datenbanktechniken untersucht.

7.1 Automatisierung der Signalauswertung

Die manuelle Neuberechnung von Signalen in der TPD führt dazu, dass abhängige Signale unter Umständen nicht in einer aktuellen Version in der TPD vorliegen. Um dies zu verhindern, wurde eine Automatisierung der Neuberechnung abhängiger TPD-Signale implementiert. Hierzu wurden Trigger für die Beschreibungstabellen der Signale erzeugt, die beim Einfügen einer neuen Signalversion in die TPD über eine erstellte PL/SQL-Prozedur die notwendigen Neuberechnungen abhängiger Signale ausführen. Da der mehrfache Aufruf der PL/SQL-Prozedur zu Problemen führt, wurde mit Hilfe einer Lockdatei ein Mechanismus implementiert, der sicherstellt, dass die verwendete Prozedur sich nur einmal in der Ausführung befindet.
Das Auflösen der direkten und indirekten Signalabhängigkeiten wird mit Hilfe einer Tiefensuche über die in der TPD zu jedem Signal gespeicherten Abhängigkeiten durchgeführt. Hierbei wird ein Abhängigkeitslevel ermittelt, um die Reihenfolge der Neuberechnung abhängiger Signale zu bestimmen, so dass doppelte Neuberechnungen eines abhängigen Signals vermieden werden können. Die eigentliche Berechnung einer neuen Signalversion erfolgt durch Aufrufe der zur Verfügung stehenden Berechnungsprogramme aus

der PL/SQL-Prozedur heraus.
Nach einer erfolgreichen Neuberechnung werden automatisch Metainformationen zu der neu erstellten Signalversion erzeugt und in die entsprechende Beschreibungstabelle der TPD eingefügt. Zur Anzeige dieser zusätzlichen Metainformationen zu einem bestimmten Signal für einen vorgegebenen Schuss in Form einer Versionshistorie wurde eine weitere PL/SQL-Prozedur implementiert. Diese fragt die benötigten Informationen aus der TPD ab und stellt sie unter Verwendung des PL/SQL-Webtoolkits als Webseite zur Verfügung.

7.2 Ergebnisse der Performance-Optimierung

Bei der Auswertung der TEXTOR-Signale spielen die Abfragezeiten der in der TPD gespeicherten Signaldaten eine wichtige Rolle. Die Abfragezeiten dauern derzeit sehr lange und erschweren deshalb ein effizientes Arbeiten mit den Signaldaten. Daher wurde untersucht, ob sich die Abfragen durch den Einsatz verschiedener Datenbanktechniken beschleunigen lassen. Hierzu wurde eine Testumgebung eingerichtet, in der Zeitmessungen an Hand verschiedener Beispielabfragen durchgeführt wurden, um verschieden Ansätze für die Performanceverbesserung zu bewerten. Neben dem Aufteilen der sehr großen Signaltabellen mit Hilfe von Tabellenpartitionierung wurde der Einsatz von Materialized Views und die Erweiterung der Indizierung der Signaltabellen untersucht. Zudem wurden Vergleiche zwischen der direkten Ausführung der Abfragen auf dem Datenbankserver und der Verwendung einer Weboberfläche zur Ausführung der Abfrage sowie zwischen dem Datenbankserver der Testumgebung und dem der Produktionsumgebung durchgeführt.
Die durchgeführten Zeitmessungen haben gezeigt, dass lediglich durch Verwendung von Materialized Views die Abfragezeiten zum Teil deutlich verbessert werden konnten. Die Partitionierung der Signaltabellen sowie die Indexerweiterung führten dagegen nur zu sehr geringen Zeitvorteilen. Da bei der Verwendung der Weboberfläche im Gegensatz zur direkten Ausführung der Datenbankabfragen sowohl die Anfragen selbst als auch die Ergebnisse der Anfragen durch ein Netzwerk transportiert werden müssen und zudem das für die Weboberfläche verwendete Java-Servlet die Anfragen verarbeiten und auswerten muss, verlangsamt die Nutzung der Weboberfläche die Durchführung der Anfragen ein wenig. Die geringen Zeitunterschiede sind jedoch unter Berücksichtigung des gewonnenen Komforts durch die Parametereingabe der Abfrage über ein Webformular und die automatische Ausführung der Datenbankabfrage durchaus annehmbar.
Die Durchführung der Beispielabfragen dauerte auf dem Produktionsserver erheblich länger als in der Testumgebung. Dies ist zum einen durch die deutlich neuere Hardware des Servers in der Testumgebung und zum anderen durch das evtl. Auftreten konkurrierende Prozesse und Anfragen aus dem Produktionsbetrieb während der Zeitmessung zu erklären. Für die bevorstehende Migration der Produktionsumgebung auf den in der Testumgebung verwendeten Server wird auf Grund der deutlichen Ergebnisse der Zeitmessungen zwischen den beiden Servern eine spürbare Performanceverbesserung erwartet.

7.3 Ausblick

Bei den Zeitmessungen wurden fünf Beispielabfragen verwendet. Diese decken jedoch nur einen Querschnitt über mögliche auftretende Abfragen ab. So wurde z.B. auf Grund der Komplexität beim Erstellen und Auswerten der Abfragen innerhalb des Java-Servlets des TPD Shot Selector keine Zeitmessung für die Schusssuche über die Merkmale 2-dimensionaler Signale durchgeführt.
Eventuell wird die Weboberfläche des TPD Shot Selector zukünftig um weitere, unter Umständen komplexe, Abfragetypen erweitert, die ebenfalls lange Abfragezeiten zur Folge haben könnten und hinsichtlich einer möglichen Performanceverbesserung untersucht werden müssen.

Der in der Testumgebung verwendete zukünftige Datenbankserver zeigte bei den Zeitmessungen deutliche Vorteile gegenüber dem in der aktuellen Produktionsumgebung eingesetzten Datenbankserver. Nach der Migration der Produktionsumgebung sind diese Zeitvorteile jedoch durch erneute Messungen zu belegen und ggf. die Auswirkung konkurrierender Anfragen aus dem Produktionsbetrieb zu untersuchen.
Neben den hier untersuchten Ansätzen stellt die verwendete Oracle®-Datenbank weitere Möglichkeiten zur Performance-Optimierung bereit. Neben der Konfiguration von Datenbankparametern wie der Aufteilung und Größer verschiedener Speicherbereiche und der Verwendung eines SQL-Tuning-Advisors gibt es weitere Möglichkeiten bei der Performanceoptimierung, die zu einer Verbesserung der Abfragezeiten führen könnten.

Die TPD enthält eine sehr große Anzahl Daten und zum Teil enorm große Tabellen, die zudem stetig wachsen. Eine mögliche Überarbeitung des Datenmodells und der Tabellenstrukturen könnte ebenfalls helfen, die Abfrage der Signaldaten zu beschleunigen und sollte zukünftig näher betrachtet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[LIT01] C. Buchal (2007)
Energie
Foschungszentrum Jülich GmbH in der Helmholtz-Gemeinschaft, Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helholtz-Gemeinschaft, Foschungszentrum Karlsruhe GmbH in der Helmholtz-Gemeinschaft
ISBN: 978-3-89336-503-6

[LIT02] J. Raeder et al. (1981)
Kontrollierte Kernfusion
Teubner-Studienbücher: Physik
ISBN: 3-519-03046-2

[LIT03] G. McCracken, P. Stott (2005)
Fusion - The Energy Of The Universe
Esevier Academic Press
ISBN: 0-12-481851-X

[LIT04] M. Korten, J.G. Krom (2006)
JDAQ, the new TEXTOR data acquisition program
Elsevier B.V.
http://www.sciencedirect.com/science/journal/09203796

[LIT05] World Wide Web Consortium (1994)
Request for Comments 1738
http://www.w3.org/Addressing/rfc1738.txt

[LIT06] Abbey, Corey, Abramson (2000)
Oracle 8i für Einsteiger
Carl Hanser Verlag München Wien
ISBN: 3-446-21432-1

[LIT07] Oracle Corporation (2002)
Professioneller Einstieg in Oracle 9i SQL

Oracle Corporation

[LIT08] E.F. Codd (1970)
A Relational Model of Data for Large Shared Data Banks
Communications of the ACM, Vol. 13, Number 6
http://www.cis.upenn.edu/ zives/03f/cis550/codd.pdf

[LIT09] V. Sander (2006)
Vorlesungsskript: Datenbankentwicklung
Fachhochschule Jülich

[LIT10] J. G. Krom (2007)
The TEXTOR Diagnostic Data Handling System
Interne Dokumentation zum TEXTOR-Experiment

[LIT11] World Wide Web Consortium (1999)
Request for Comments 2616
http://www.w3.org/Protocols/rfc2616/rfc2616.html

[LIT12] Oracle Corporation (2002)
Oracle Database 10g: Administration Workshop I, Band 1
Oracle Corporation

[LIT13] G. Türscher (1997)
PL/SQL
Springer-Verlag Berlin Heidelberg
ISBN: 3-540-61319-6

[LIT14] C. Ostrowski, B. Brown (2005)
Oracle Application Server 10g Web Development
McGraw-Hill
ISBN: 0-07-225511-0

[LIT15] Sam R. Alapti (2005)
Expert Oracle Database 10g Administration
Apress
ISBN-13 (pkb): 978-1-59059-451-3

[LIT16] R. Hlusiak (2008)
Bachelorarbeit: Entwicklung eines Cutoff-Analyse-Tools für TEXTOR-Daten

[WEB1] IPP (2008)
Fusion 21 - Plasmaheizung
Max-Planck-Institut für Plasmaphysik
`http://www.ipp.mpg.de/ippcms/de/pr/fusion21/plasmaheizung/index.html`

[WEB2] IEF-4 (2003)
TEXTOR
Forschungszentrum Jülich GmbH in der Helmholtz-Gemeinschaft - Stand 2003
`http://www.fz-juelich.de/ief/ief-4/textor/`

[WEB3] Wikipedia (2008)
CAMAC
Wikimedia Foundation Inc.
`http://de.wikipedia.org/wiki/CAMAC`

Jül-4309
September 2009
ISSN 0944-2952